AMath 483/583 — Lecture 28

Outline:

- Numba and autojit
- Binary vs. ASCII output
- Review / take away messages

See also:

- Numba
- `$UWHPSC/codes/io`

Just-in-time compilers for Python

Standard implementation of Python as interpreted language.

Importing `mymodule.py` creates `mymodule.pyc`, which is Bytecode (portable code or pcode):
  One-byte operators with operands,
  Interpreted by software at runtime.

Runs much slower than compiled code that is machine-specific instructions.

Just-in-time compilers for Python

Standard implementation of Python as interpreted language.

Importing `mymodule.py` creates `mymodule.pyc`, which is Bytecode (portable code or pcode):
  One-byte operators with operands,
  Interpreted by software at runtime.

Runs much slower than compiled code that is machine-specific instructions.

Just-in -time (JIT) compilation: Converts bytecode at runtime into native machine code.

Can sometimes run faster than pre-compiled code.

Just-in-time compilers for Python

Examples:

- PyPy — alternative implementation of Python
- numba — compiles decorated code to LLVM
  (formerly Low Level Virtual Machine,
  compiler infrastructure)

  Included in the Anaconda Python distribution

Numba — autojit decorator

```
In [1]: def loopsum(n):
            x = 0
            for i in range(n):
                x = x + i
```

```
In [2]: %timeit loopsum(10000)
```

```
1000 loops, best of 3: 495 us per loop
```

Numba — autojit decorator

```
In [1]: def loopsum(n):
            x = 0
            for i in range(n):
                x = x + i
```

```
In [2]: %timeit loopsum(10000)

1000 loops, best of 3: 495 us per loop
```

```
In [3]: from numba import autojit
```

```
In [4]: @autojit
        def loopsum2(n):
            x = 0
            for i in range(n):
                x = x + i
```

```
In [5]: %timeit loopsum2(10000)

1000000 loops, best of 3: 1.5 us per loop
```

ASCII vs. binary output

Often need to write out a large array of floats with full precision.

For example, one solution value on 3d grid ...

```
do i=1,n
    do j=1,n
        do k=1,n
            write(21,'(e24.16)') u(i,j,k)
        enddo; enddo; enddo
```

How much disk space does this take?

ASCII vs. binary output

Often need to write out a large array of floats with full precision.

For example, one solution value on 3d grid ...

```
do i=1,n
    do j=1,n
        do k=1,n
            write(21,'(e24.16)') u(i,j,k)
        enddo; enddo; enddo
```

How much disk space does this take?

A single number such as `0.4000000000000000E+01` has 24 ASCII characters $\Longrightarrow$ 24 bytes per value.

Total $24n^3$ bytes. E.g. $100 \times 100 \times 100$ grid: $n = 100 \implies 24$ MB.

ASCII vs. binary output

Often need to write out a large array of floats with full precision.

For example, one solution value on 3d grid ...

```
do i=1,n
    do j=1,n
        do k=1,n
            write(21,'(e24.16)') u(i,j,k)
        enddo; enddo; enddo
```

How much disk space does this take?

A single number such as `0.4000000000000000E+01`
has 24 ASCII characters $\implies$ 24 bytes per value.

Total $24n^3$ bytes. E.g. $100 \times 100 \times 100$ grid: $n = 100 \implies 24$ MB.

Note: In memory storing one 8-byte float takes only 8 bytes.
($n = 100 \implies 8$MB.) ASCII takes $3\times$ the space.

ASCII vs. binary output

Often need to write out a large array of floats with full precision.

For example, one solution value on 3d grid ...

```
do i=1,n
    do j=1,n
        do k=1,n
            write(21,'(e24.16)') u(i,j,k)
        enddo; enddo; enddo
```

How much disk space does this take?

A single number such as `0.4000000000000000E+01`
has 24 ASCII characters $\implies$ 24 bytes per value.

Total $24n^3$ bytes. E.g. $100 \times 100 \times 100$ grid: $n = 100 \implies 24$ MB.

Note: In memory storing one 8-byte float takes only 8 bytes.
($n = 100 \implies 8$MB.) ASCII takes $3\times$ the space.
Also takes additional time to convert to ASCII,
$\approx 10\times$ slower to write ASCII than dumping binary.

Binary output in Fortran

Can use unformatted write in Fortran:

```
! $UWHPSC/codes/io/binwrite.f90

open(unit=20, file="u.bin", form="unformatted", &
     status="unknown", access="stream")

do j=1,100
    do i=1,500
        u(i,j) = real(m*(j-1) + i, kind=8)
    enddo
enddo

write(20) u   ! writes entire array in binary
close(20)
```

```
-----------------------------------------------
$ ls -l
-rw-r--r--  1 rjl  staff   400000 Jun  6 20:09 u.bin
-rw-r--r--  1 rjl  staff  1200000 Jun  6 20:09 u.txt
```

Binary output in Fortran

Can use unformatted write in Fortran:

```
! $UWHPSC/codes/io/binwrite.f90

open(unit=20, file="u.bin", form="unformatted", &
     status="unknown", access="stream")

do j=1,100
    do i=1,500
        u(i,j) = real(m*(j-1) + i, kind=8)
    enddo
enddo

write(20) u    ! writes entire array in binary
close(20)
```

```
---------------------------------------------
$ ls -l
-rw-r--r--  1 rjl  staff   400000 Jun  6 20:09 u.bin
-rw-r--r--  1 rjl  staff  1200000 Jun  6 20:09 u.txt
```

The resulting binary file `u.bin` cannot be edited directly.

But we can read it into Python...

Reading binary data files in Python

To recover `U` array of dimension $m \times n$ in Python:

```
# $UWHPSC/codes/io/binread.py

import numpy as np

file = open('u.bin', 'rb')
uvec = np.fromfile(file, dtype=np.float64)

m,n = np.loadtxt('mn.txt',dtype=int)

# now use Fortran ordering to fill u by columns:
u = uvec.reshape((m,n),order='F')
```

Other options for binary data

Binary formats that contain a lot of metadata...

Hierarchical Data Format: HDF, HDF4, HDF5

HDF5 file structure includes two major types of object:

- Datasets: multidimensional arrays of a homogenous type
- Groups: container structures for datasets and other groups

See also: h5py, PyTables

Other options for binary data

Binary formats that contain a lot of metadata...

Hierarchical Data Format: HDF, HDF4, HDF5

HDF5 file structure includes two major types of object:

- Datasets: multidimensional arrays of a homogenous type
- Groups: container structures for datasets and other groups

See also: h5py, PyTables

NetCDF (Network Common Data Form): Built on top of HDF5.

See also ncdump, netcdf4-python

Summary, take away messages...

- Version control — git
  Use for all your projects, collaborations, ...
  Consider contributing to open source projects
      Submit a pull request

Summary, take away messages...

- Version control — git
  Use for all your projects, collaborations, ...
  Consider contributing to open source projects
    Submit a pull request

- Python, NumPy, SciPy, matplotlib, IPython
  Quickly trying out new ideas, optimize later
  Graphics and visualization
  Scripting to guide big computations
  Combining codes from different languages
  Many capabilities not seen in class, e.g.
    Manipulating text files, regular expressions,
    building web interfaces

Summary, take away messages...

- Fortran 90
  Compiled language
  Tightly constrained but can run very fast
  Native multi-dimensional arrays

Summary, take away messages...

- Fortran 90
  Compiled language
  Tightly constrained but can run very fast
  Native multi-dimensional arrays

- Makefiles
  Dependency checking
  Often used for building software

Summary, take away messages...

- Fortran 90
  Compiled language
  Tightly constrained but can run very fast
  Native multi-dimensional arrays

- Makefiles
  Dependency checking
  Often used for building software

- Debugging code
  Unit tests, nose module
  Print statements, pdb, gdb

Summary, take away messages...

- Fortran 90
  Compiled language
  Tightly constrained but can run very fast
  Native multi-dimensional arrays

- Makefiles
  Dependency checking
  Often used for building software

- Debugging code
  Unit tests, nose module
  Print statements, pdb, gdb

- Memory hierachy, cache considerations
  Consider layout of arrays in memory
  Aim for spatial and temporal locality

Summary, take away messages...

- Parallel computing
  Increasingly necessary for all computing
  Amdahl's law —
  inherently sequential code limits parallelization
  Weak vs. strong scaling
  Fine grain vs. coarse grain parallelism
  Load balancing

Summary, take away messages...

- Parallel computing
  Increasingly necessary for all computing
  Amdahl's law —
  inherently sequential code limits parallelization
  Weak vs. strong scaling
  Fine grain vs. coarse grain parallelism
  Load balancing

- OpenMP
  Assumes shared memory
  Often very easy to add to existing codes
  Need to worry about shared/private variables,
  race conditions

Summary, take away messages...

- MPI — Message Passing Interface
  Always assumes distributed memory
  Sharing data requires message passing
  SPMD: Single Program Multiple Data
  Entire program run by each process
      But different processes may take different branches

Summary, take away messages...

- MPI — Message Passing Interface
  Always assumes distributed memory
  Sharing data requires message passing
  SPMD: Single Program Multiple Data
  Entire program run by each process
  But different processes may take different branches

- Computer arithmetic
  Floating point number representation, 4 byte vs. 8 byte
  IEEE standards
  Reproducibility still difficult in parallel
  Relative error and precision possible
  Condition number of problem / stability of algorithm

Summary, take away messages...

- Linear algebra
  Matrix norms and condition number of $Ax = b$
  LAPACK, BLAS — optimized code
  Iterative methods for large sparse system
  Poisson problems: $u_{xx} = f(x) \implies$ tridiagonal
  Two-dimensional Poisson problem $u_{xx} + u_{yy} = f(x, y)$

Summary, take away messages...

- Linear algebra
  Matrix norms and condition number of $Ax = b$
  LAPACK, BLAS — optimized code
  Iterative methods for large sparse system
  Poisson problems: $u_{xx} = f(x) \implies$ tridiagonal
  Two-dimensional Poisson problem $u_{xx} + u_{yy} = f(x, y)$

- Quadrature methods / numerical integration
  Midpoint, Trapezoid, Simpson Rules
  Adaptive Quadrature / Load balancing
  Monte Carlo methods in high dimensions

Summary, take away messages...

- Linear algebra
  Matrix norms and condition number of $Ax = b$
  LAPACK, BLAS — optimized code
  Iterative methods for large sparse system
  Poisson problems: $u_{xx} = f(x) \implies$ tridiagonal
  Two-dimensional Poisson problem $u_{xx} + u_{yy} = f(x, y)$

- Quadrature methods / numerical integration
  Midpoint, Trapezoid, Simpson Rules
  Adaptive Quadrature / Load balancing
  Monte Carlo methods in high dimensions

- Monte Carlo methods
  Pseudo Random Number Generation
  Use of seed for reproducibility
  Random walks

Happy Computing!

Happy Computing!

Thanks for participating.

Happy Computing!

Thanks for participating.

Thanks to TAs: Scott Moe and Susie Sargsyan

Happy Computing!

Thanks for participating.

Thanks to TAs: Scott Moe and Susie Sargsyan

Office hours: See discussion board.

Happy Computing!

Thanks for participating.

Thanks to TAs: Scott Moe and Susie Sargsyan

Office hours: See discussion board.

Have a great summer!