Serial Optimization Tips

HPC BOOTCAMP

- Optimization, bag-of-tasks, MPI, OpenMP
- Week of June 14
- Free!
- Goodies provided for morning and afternoon breaks
- Python is taught except for OpenMP

In Nearly All Cases The Tradeoff Is:

- Memory versus speed
  - More memory usage=more speed UNTIL you use enough memory that you slow down (or crash) the entire system
  - More memory usage=fewer options for running (less of a problem nowadays)
  - But if you do use enough memory to slow down the system it will be MUCH slower.
    - Please try to avoid 4000x4000 arrays!

For All Languages:

- Do not recompute
  - We see many codes where programmers recompute the same quantity over and over
  - If it's a scalar like pi/180.0 this isn't horrible, but it's not great either, and it's not necessary! Compute it first and store it.
  - Do not recompute *anything* in a big loop that you can precompute and store.
  - Do not recompute information already accumulated even if you must introduce a new array (unless it's huge).

Loop Efficiency

- Do not recompute array accesses. Make a variable

```
do j=1,10
   do i=1,10
      Something=r(j)*sin(z(i-j+1))*x(i,j)
   Enddo
enddo
```

Better

```
do j=1,10
   Rj=r(j)
   j1=1-j
   do i=1,10
      Something=Rj*sin(z(i+j1))*x(i,j)
   enddo
enddo
```

More Loop Efficiency

- Avoid conditionals in inner loops
- Sometimes it's best to go through large multidimensional arrays as if they were linear (which they are in memory).
- Or just use an array operation and trust the compiler/interpeter

Reduce Math Functions

- Fast:
  - Add/subtract/multipy (fma: fused multiply add is usually a hardware instruction also, i.e. a=a+b*c)
- Slow:
  - Division
- Really slow:
  - Math functions especially power (to noninteger in Fortran, to anything in other languages) and square root and such.

Fortran

- Make sure loops are in column-major order (right to left)

```
do j=1,10000
   do i=1,10000
      z(i,j)=a(i,j)+b
   enddo
enddo
```

- This is for *stride 1 memory access* and *cache efficiency*
- Opposite for row-major order languages (C/C++ etc.)

Python

- Avoid dynamically resized lists, or anything that changes dynamically during the run after it is initialized
  - OK to allocate/size at runtime, but once the size is fixed, allocate it and keep it the same size
  - NumPy array versus list
  - Frozenset versus set
- Avoid `for` loops. Use almost anything else:
  - List comprehension
  - Iterator

Python Example

```
for i in range(1,n-1):
    for j in range(1,n-1):
        u[i,j]=0.25*(u[i+1,j]+u[i-1,j]+u[i,j\
+1]+u[i,j-1])
```

- Can be replaced by
- `u[1:-1,1:-1]=0.25*(u[2:,1:-1]+u[-2:1:-1]+\`
  `u[1:-1,2:]+u[1:-1,-2:]`
- This can result in a speedup of up to 250x.

Python Tips and Tricks

- Don’t concatenate strings, use the join method instead (or use print to format a string)
- Use map rather than a loop to apply a function to a sequence

- http://wiki.python.org/moin/PythonSpeed/PerformanceTips