Some quick tips to speed up your analysis

• Optimizing code
• Compilation
What’s wrong with this?
Hint: there are two speed issues, and two style issues

# Compute the vector product of the two velocities
i = 10
j = 0
while j < i+1:
    k = i + 3
    s[j] += k*j
    j += 1
# Add a displacement to the distance array

```python
interval = 10
limit = interval + 1
scale = interval + 3
j = 0
while j < limit:
    distance[j] += scale*j
    j += 1
```
Vectorization
This means taking loops over arrays and splitting up the operations onto multiple processors

- C++
  - Just add “-O3” to the compilation command; e.g.
    ```
g++ -O3 myprog.cxx -o myprog
    ```

- Python
  - Python vectorization involves full or partial conversion of Python into C code; e.g., Cython and Numba, which are Python compilers (and are available on the Nevis particle-physics systems).
    - Cython: [https://cython.org/](https://cython.org/)
  - With numpy, there may be a simpler way...
# Speeding up numpy

When you’re using a Python loop with numpy arrays, try to find a numpy function that does the same thing.

```python
import math, numpy as np

data = np.ones(shape=(1000,1000),dtype=np.float)
sqdata = data

# This is slow:
for i in range(1000):
    for j in range(1000):
        data[i][j] *= 2.0
        sqdata[i][j] = math.sqrt(data[i][j])

# This is fast. We’re using numpy’s definition of “*” and sqrt.
data *= 2.0
sqdata = np.sqrt(data)
```