Some quick tips to speed up your analysis

- Optimizing code
- Vectorization
- RDataFrame

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
while j < i+1:
  k = i + 3
  s[j] += k*j
  j += 1
```

Better

```
# Add a displacement to the distance array
interval = 10
limit = userNumber + 1
scale = interval + 3
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).
    - Numba: http://numba.pydata.org/
    - Cython: 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.

```
import math, numpy as np

# Create a couple of 1000x1000 2D arrays filled with 1's
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)
```

RDataFrame

An RDataFrame lets you do column-wise operations on an ntuple or TTree, instead of the row-wise operations you've done so far in the tutorial.

You can do ntuple analysis in a more "spreadsheet-like" way.

RDataFrame is a ROOT class. You can look it up in the ROOT web site (as you did with THD). It works with both C++ and Python.

Branches -->

<-- Entries

Row	event	ebeam	px	py	pz
0	0	150.14	14.33	-4.02	143.54
1	1	149.79	0.05	-1.37	148.60
2	2	150.16	4.01	3.89	145.69
3	3	150.14	1.46	4.66	146.71
4	4	149.94	-10.34	11.07	148.33
5	5	150.18	17.08	-12.14	143.10
6	6	150.02	5.19	7.79	148.59
7	7	150.05	7.55	-7.43	144.45
8	8	150.07	0.23	-0.02	147.78
9	9	149.96	1.21	7.27	146.99
10	10	149.92	5.35	3.98	140.70
11	11	149.88	-4.63	-0.08	147.91