From simulation to knowledge

Speeding up your data analysis
From simulation to knowledge

Speeding up your data analysis

Where does time go?

- CPU speed is app. 0.1 ns (@ 3.3 GHz)
- Cache memory is app. 5 ns
- DRAM memory access is app. 100 ns
- SSD access is app. 5 ms
- HDD access is app. 5 ms

Most important thing to take home:
Processing speed is all about data movement, processing is mostly free.
Where does time go?

CPU speed is app 0.1 ns (@3.3 GHz)
Cache memory is app 5 ns
Main memory access is app 30 ns
SSD access is app 100 ns
HDD access is app 5 ms

Most important thing to take home:
Processing speed is all above data-movement processing is mostly for free
Approaches to speed up analysis

1) Try to read data only once
2) Read only the data you need
3) Parallelize your analysis
Parallelize I

Use vectorized code

```python
def sum_wrong(data):
    result = numpy.zeros((x,y,z))
    for t_i in range(t):
        for z_i in range(z):
            for y_i in range(y):
                for x_i in range(x):
                    result[x_i, y_i, z_i] += data[t_i, x_i, y_i, z_i]
    return result
```

134.12 sec

```python
def sum_better(data):
    result = numpy.zeros((x,y,z))
    for t_i in range(t):
        result += data[t_i]
    return result
```

0.16 sec

```python
def sum_right(data):
    return data.sum(axis=0)
```

0.15 sec
Use vectorized code

```python
def sum_wrong(data):
    result = numpy.zeros((x, y, z))
    for t_i in range(t):
        for z_i in range(z):
            for y_i in range(y):
                for x_i in range(x):
                    result[x_i, y_i, z_i] += data[t_i, x_i, y_i, z_i]
    return result
```

```
def sum_better(data):
    result = numpy.zeros((x, y, z))
    for t_i in range(t):
        result += data[t_i]
    return result
```
return result

def sum_better(data):
    result = numpy.zeros(((x, y, z)))
    for t_i in range(t):
        result += data[t_i]
    return result

def sum_right(data):
return result

def sum_right(data):
    return data.sum(axis=0)
134.12 sec

0.16 sec

0.15 sec
Parallelize II

Make your IO parallel

Map - Reduce
Make your IO parallel
Map - Reduce
[Call me Ishmael. Some years ago—never mind how long precisely—having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of divesting off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral principle to prevent me from deliberately stepping into the street, and meeting with the hodiernal knocking of people's hats off—then, I account it high time to get to sea as soon as I can. Then is my substitute for pistol and ball. With a philosophical flourish Cato throws himself upon his sword; I quietly take to the ship. There is nothing surprising in this. If they but knew it, almost all men in their degree, some time or other, cherish very nearly the same feelings towards the ocean with me.]
Call me Ishmael. Some years ago—never mind how—long precisely—having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral principle to prevent me from deliberately stepping into the street, and methodically knocking people's hats off—then, I account it high time to get to sea as soon as I can. This is my substitute for pistol and ball. With a philosophical flourish Cato throws himself upon his sword; I quietly take to the ship. There is nothing surprising in this. If they but knew it, almost all men in their degree, some time or other, cherish very nearly the same feelings towards the ocean with.
Map Reduce on netCDF

```python
import numpy

x = 100; y = 100; z = 100; t = 100; p = 3

def choose_temperatures(data):
    return data[1,:,:,:,:]

tensor_sum = lambda a,b: a + b

data = numpy.random.random((t, p, x, y, z))

temperatures = map(choose_temperatures, data)
result = reduce(tensor_sum, temperatures)
```
import numpy

x = 100; y = 100; z = 100; t = 100; p = 3

def choose_temperatures(data):
    return data[1,:,:,:,:]
```python
import numpy

x = 100; y = 100; z = 100; t = 100; p = 3

def choose_temperatures(data):
    return data[1,:,:,:,:]

tensor_sum = lambda a,b: a + b

data = numpy.random.random((t, p, x, y, z))

temperatures = map(choose_temperatures, data)
result = reduce(tensor_sum, temperatures)
```
SOFA and BDAE

class ExampleNetCDFCollection(NetCDFDatasetCollection):
    def get_operations(self):
        return []

    def get_dataset_type(self, identifier):
        return ExampleNumberDataset(name=identifier)

    def get_identifiers(self):
        return ['pressure', 'temperature', 'humidity']

class ExampleNumberDataset(NumpyArrayDataset):
    def get_operations(self):
        return [OperationContext.by(self, 'unique', {})]

    def get_map_functions(self):
        return NumpyArrayDataset.get_map_function()

    def get_reduce_functions(self):
        return module_binder(numpy, reduce_functions)

    def preprocess(self, self, data_ref):
        return data_ref

    def next_entry(self, self, data):
```python
class ExampleNetCDFCollection(NetCDFDatasetCollection):
    def get_operations(self):
        return []

    def get_dataset_type(self, self, identifier):
        return ExampleNumberDataset(name=identifier)

    def get_identifiers(self, self):
        return ['pressure', 'temperature', 'humidity']
```
from bdae.templates.import_utils import reduce_function_binder, module_binder
from bdae.templates.number_dataset import NumpyArrayDataset
from sofa.foundation.operation import OperationContext


class ExampleNumberDataset(NumpyArrayDataset):
    def get_operations(self):
        return [
            OperationContext.by(self, 'unit sum', '[sink, sum]')
        ]

    def get_map_functions(self):
        return NumpyArrayDataset.get_map_functions(self) + [sink]

    def get_reduce_functions(self):
        return module_binder(numpy, reduce_function_binder, ['sum'])

    def preprocess(self, data_ref):
        return data_ref

    def next_entry(self, data):
        for d in data:
            yield d

    def sink(self, blocks, args):
        return blocks, args
scientist.submit_job("temperature", "unit sum", None, callback=result_callback)
CT Reconstruction in BDAE
Summary

Reading data is the most expensive operation you there is

Much analysis is trivially parallelized - but use tools - do not write your own parallel programs

Big data tools are well suited for data analysis - but traditional tools like Hadoop suffer from the residual problem
GC-503
MONTAÑA
LA DATA
From simulation to knowledge

Speeding up your data analysis

Where does time go?

CPU speed: 1 ms (3.3 GHz)
Cache memory: 1 ms
Main memory access: 30 ms
SSD access: 500 ms
HDD access: 5000 ms

Most important thing to take home:
Processing speed is all about data movement.
Momentum is mostly for free.