Chapel's Home in the New Landscape of Scientific Frameworks

(and what it can learn from the neighbours)

Jonathan Dursi
Senior Research Associate
Centre for Computational Medicine
The Hospital for Sick Children
https://github.com/ljdursi/CHIUW2017

Who Am I?

Old HPC Hand...

Ex-astrophysicist turned large-scale computing.

- Large-scale high-speed adaptive reactive fluid fluids
- DOE ASCI Center at Chicago
  - ASCI Red
  - ASCI Blue
  - ASCI White
- FORTRAN, MPI, Oct-tree regular adaptive mesh
- Joined HPC centre after postdoc
- Worked with researchers on wide variety of problems

Who Am I?

Old HPC Hand...

Living in Exciting Times...

Started my career (c1995-2005) when large scale scientific computing was:

- ~20 years of stability
- Bunch of x86, MPI, ethernet or infiniband
- No one outside of academia was much doing big number/data crunching
- Pretty stable set of problems

Now found myself thrust into the most exciting time in scientific computing maybe ever.

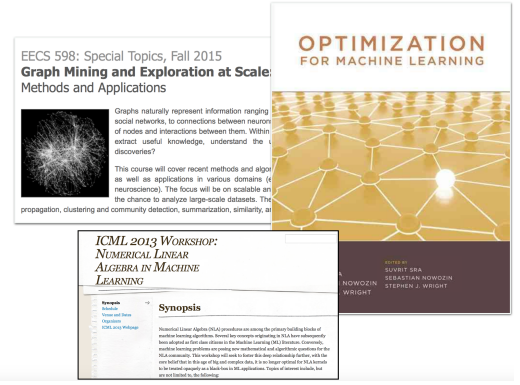
Who Am I?

Old HPC Hand...

Living in Exciting Times..

New Communities Make things Exciting!

- Internet-scale companies (Yahoo!, Google)
  - Very large-scale image processing
  - Machine learning:
  - Sparse linear algebra
  - k-d trees
  - Calculations on unstructured meshes (graphs)
  - Numerical optimization
- Genomics
  - Lots of data
  - Lots of analysis challenges
  - Large graphs for assembly, analysis
  - Large tables for statistics
- Building new frameworks



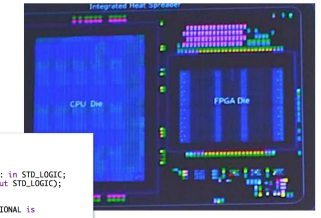
Who Am I?

Old HPC Hand...

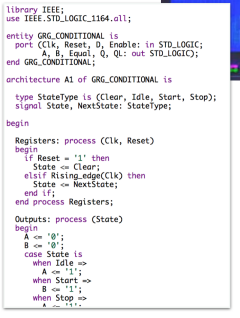
Living in Exciting Times...

New Hardware Makes things Exciting!

- Now:
  - Large numbers of cores per socket
  - GPUs/Phis
- Next few years:
  - FPGA (Intel: Broadwell + Arria 10, shipping 2017)
  - Non-volatile Memory (external memory/out-of-core algorithms)



http://intel.com



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity GRG_CONDITIONAL is
  port (Clk, Reset, D, Enable: in STD_LOGIC;
        A, B, Equal, Q, QL: out STD_LOGIC);
end GRG_CONDITIONAL;

architecture A1 of GRG_CONDITIONAL is

  type StateType is (Clear, Idle, Start, Stop);
  signal State, NextState: StateType;

begin

  Registers: process (Clk, Reset)
  begin
    if Reset = '1' then
      State <= Clear;
    elsif Rising_edge(Clk) then
      State <= NextState;
    end if;
  end process Registers;

  Outputs: process (State)
  begin
    A <= '0';
    B <= '0';
    case State is
      when Idle =>
        A <= '1';
      when Start =>
        B <= '1';
      when Stop =>
```

http://www.edaplayground.com

Richer Scientific Problems Make things Exciting!

- New science demands: cutting edge models are more complex. An Astro example:
  - 80s - gravity only N-body, galaxy-scale
  - 90s - N-body, cosmological
  - 00s - Hydrodynamics, cosmological
  - 10s - Hydrodynamics + rad transport + cosmological

Who Am I?

Old HPC Hand...

Living in Exciting Times...

Gone Into Genomics

Started looking into Genomics in ~2013:

- Large computing needs
- Very interesting algorithmic challenges
- HPCer to the rescue, right?

Made move in 2014

- Ontario Institute for Cancer Research
- Working with Jared Simpson, author of ABySS (amongst other things)
    - First open-source human-scale de novo genome assembler
    - MPI-based

Who Am I?

Old HPC Hand...

Living in Exciting Times...

Gone Into Genomics

Started looking into Genomics in ~2013:

- Large computing needs
- Very interesting algorithmic challenges
- HPCer to the rescue, right?

Made move in 2014

- Ontario Institute for Cancer Research
- Working with Jared Simpson, author of ABySS (amongst other things)
    - First open-source human-scale de novo genome assembler
    - MPI-based
- ABySS 2.0 just came out, with a new non-MPI mode

Who Am I?

Old HPC Hand...

Living in Exciting Times...

Gone Into Genomics

In the meantime, one of the de-facto standards for genome analysis, GATK, has just announced that version 4 will support distributed cluster computing — using Apache Spark.

Outline

- A survey of the evolving landscape of Big Computing frameworks
- A tour of some common big-data computing problems
  - Genomics and otherwise
  - Not so different from complex simulations
- A tour of programming models to tackle them, and lessons we can learn
  - R
  - Spark
  - Dask
  - Distributed TensorFlow
  - Coarray Fortran
  - Julia
  - Rust, Swift
- Where Chapel is, and what nearby territories look fertile

Outline

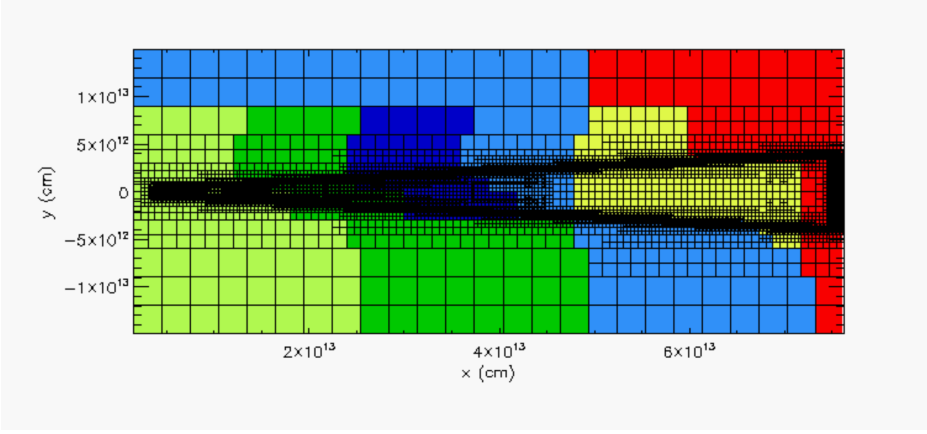
With problems in mind:

Grid PDES

My perspective is based on the sorts of problems I've worked on.

Will have those in mind when looking at languages and techniques.

Started with high-speed reactive fluid flows, either fixed grid (structured or unstructured) or block-structured adaptive:



Outline

With problems in mind:

Grid PDEs

Substring operations

(Much) more recently, working with genomics sequence data.

Assembly:

- Have small fragments of sequence, must generate whole
- Graph methods (de Bruijn or overlap graph)
- Find maximal unambiguous paths through the graph

Or may have an assembled graph genome and try to find best match for given observed subsequence

Or just count observed subsequences

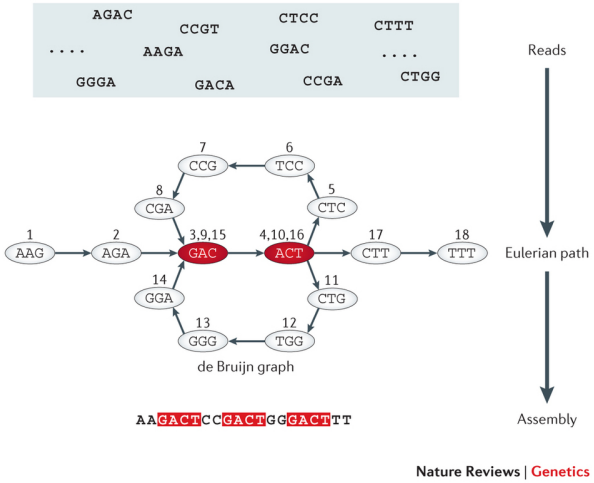


Figure from Nature Review Genetics

Outline

With problems in mind:

Grid PDEs

Substring operations

Large statistical analyses

Or just large biostatistical analyses:

Closest to my current day job (distributed analysis of private genomics data sets)

Imagine RNA sequence expression data:

- 100m fragments of sequence (imperfect sampling)
- Assigned to particular RNA transcripts
- Find out if transcripts are differentially expressed between case and condition

Now do that for multiple tissue types, large population...

And start correlating with other information (DNA variants, clinical data, phenotypic data,...)

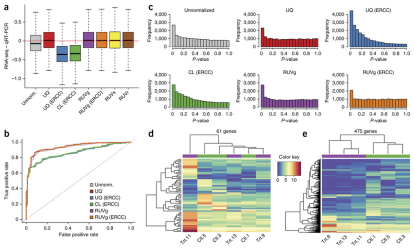
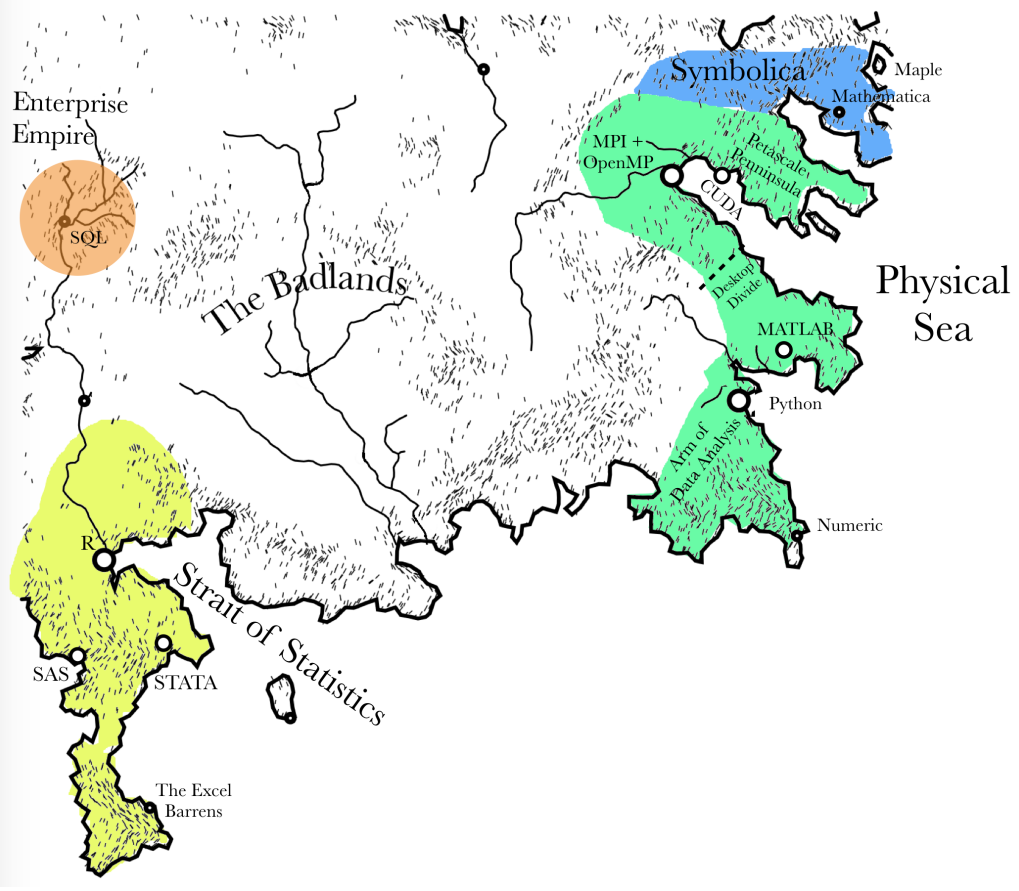


Figure from Nature

The Lay Of The Land: 2002, 2007, and 2017

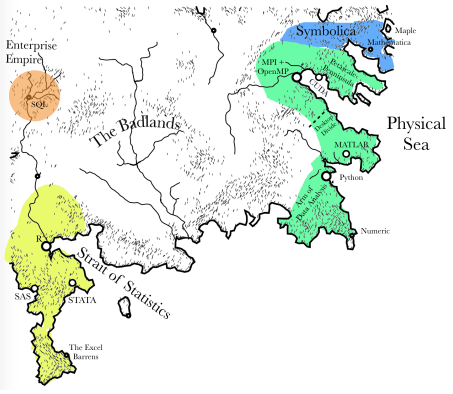
Ye Olde Entire Scientific Computing Worlde, c. 2002



(map from http://mewo2.com/notes/terrain/)

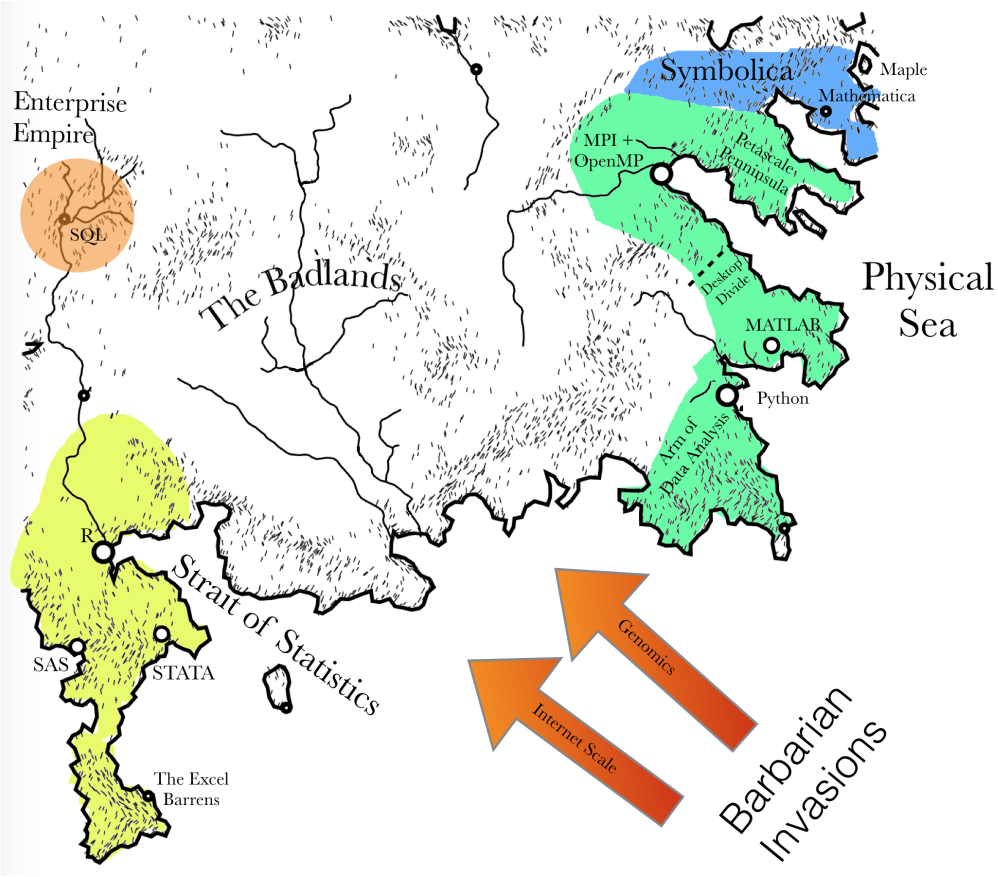
Ye Olde Entire Scientific Computing Worlde, c. 2002

It was a simpler time:



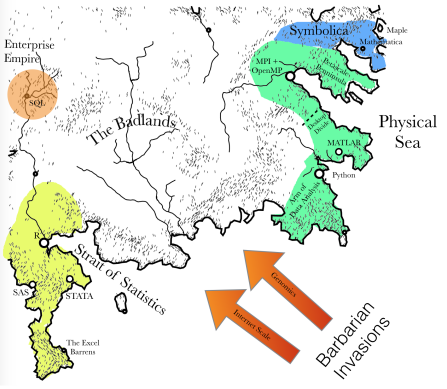
- Statistial Computing largely the domain of the social sciences, some experimental sciences
  - R was beginning to be quite popular
- Physical scientists working with Big Iron or workstations, performing simulation or analysis of comparitively regular data sets
  - FORTRAN/C/C++(?) + MPI + OpenMP
  - FORTRAN/C/C++(?)
  - MATLAB, IDL
  - Python (Numeric)
- Not a lot of SQL/database work in traditional technical computing, but communications up and downstream w/ statistical computing
- Maybe infrequent ferry service between statistical computing and MATLAB communities

And Then They Came, c. 2007



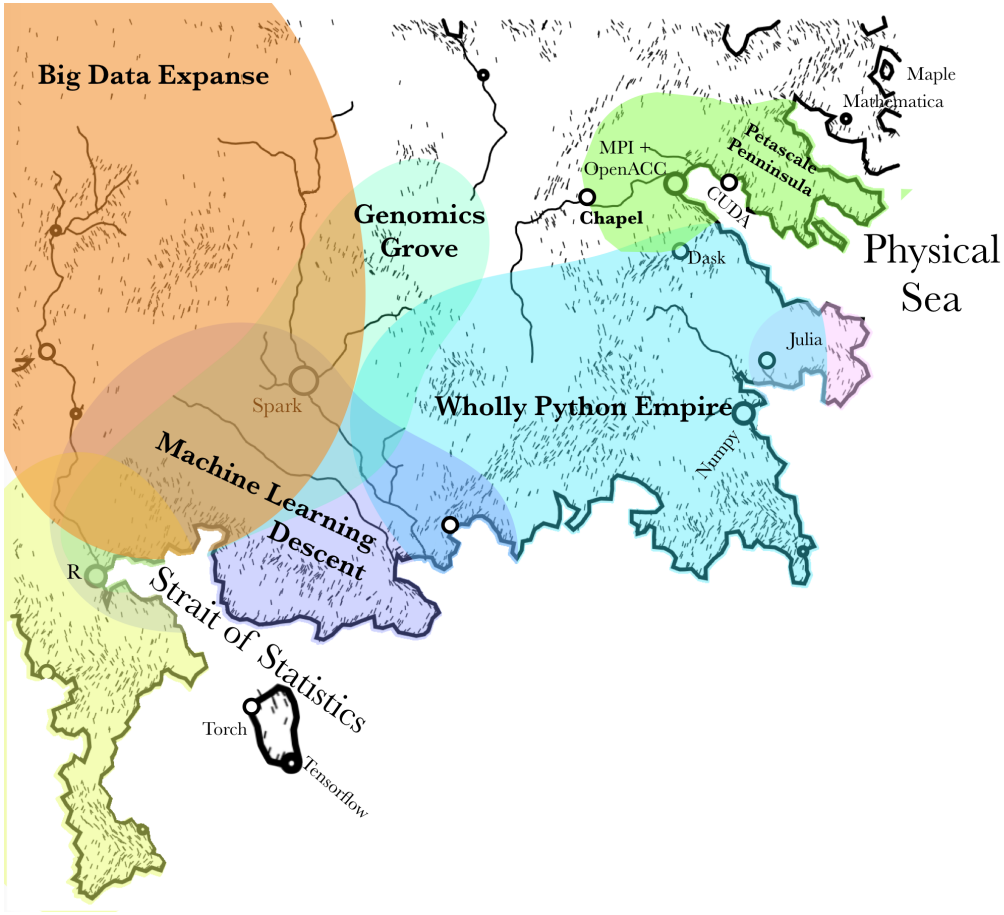
And Then They Came, c. 2007

Widespread adption of computing and networking brought *data*, and lot of it.



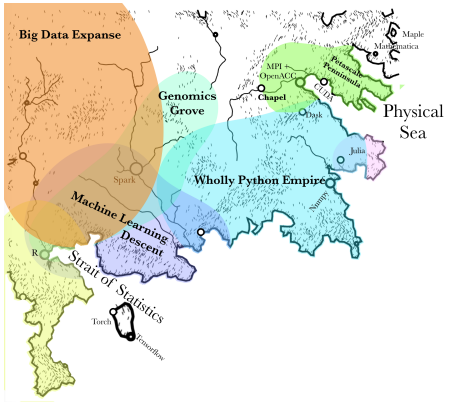
- "Internet-scale" companies were the first businesses to try taking advantage of all their data, but others soon followed
    - Hadoop, HDFS spawned an entire ecosystem
- In the sciences, genomics was in the right place at right time
    - Success of Human Genome Project in 2003
    - High-throughput sequencing technologies becoming available
    - Lots and lots of data - but how to process it?

The Present Day, 2017



The Present Day, 2017

- The newcomers started with some of their own tools (Hadoop, HDFS)

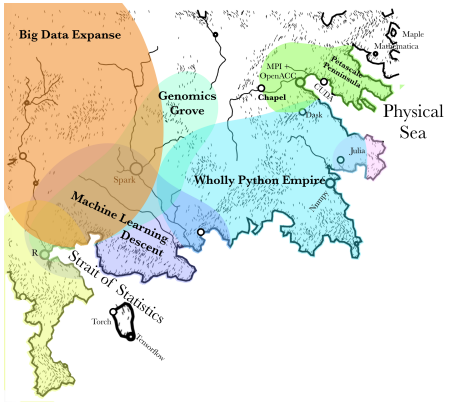


- (Some of) the data-analysis handling communities jumped at the chance to start working with the data-intensive newcomers
  - Similar needs, interests
  - Python on the general computing and physical sciences side
  - R on the statistics/Machine Learning (neé data mining) side
- The simulation science communities, which makes up most of traditional HPC, were more skeptical
  - Needs seemed very different
  - Very different terminology
  - Initial tools (Hadoop Map-Reduce) were all out of core, calculations very simple (analytics)
  - Still not a lot of overlap

The Present Day, 2017

Will argue that they are not so different, and there's a lot to learn (on both sides) across the data science/simulation science divide

- Simulations are getting more complex, dynamic
- Big data problems have long been in-memory, increasingly compute intensive
- Moving towards each other in fits and starts



Big Data Problems

Big Data problems same as HPC, if in different context

- Large scale network problems
  - Graph operations
- Similarity computations, clustering, optimization,
  - Linear algebra
  - Tree methods
- Time series
  - FFTs, smoothing, ...

Big Data Problems

Linear algebra

Almost any sort of numeric computation requires linear algebra.

In many big-data applications, the linear algebra is *extremely* sparse and unstructured; say doing similarity calculations of documents, using a bag-of-words model.

If looking at ngrams, cardinality can be enormous, no real pattern to sparsity

	abstract	galaxy	expression	gene	supernova	dna	star
a=	1	1			1		1
g=	1		1	1		1	

$$
\begin{aligned}
S_{a,g} &= \frac{\mathbf{w_a} \cdot \mathbf{w_g}}{||w_a|| \cdot ||w_g||} \\
&= \frac{1}{2 \cdot 2} \\
&= \frac{1}{4}
\end{aligned}
$$

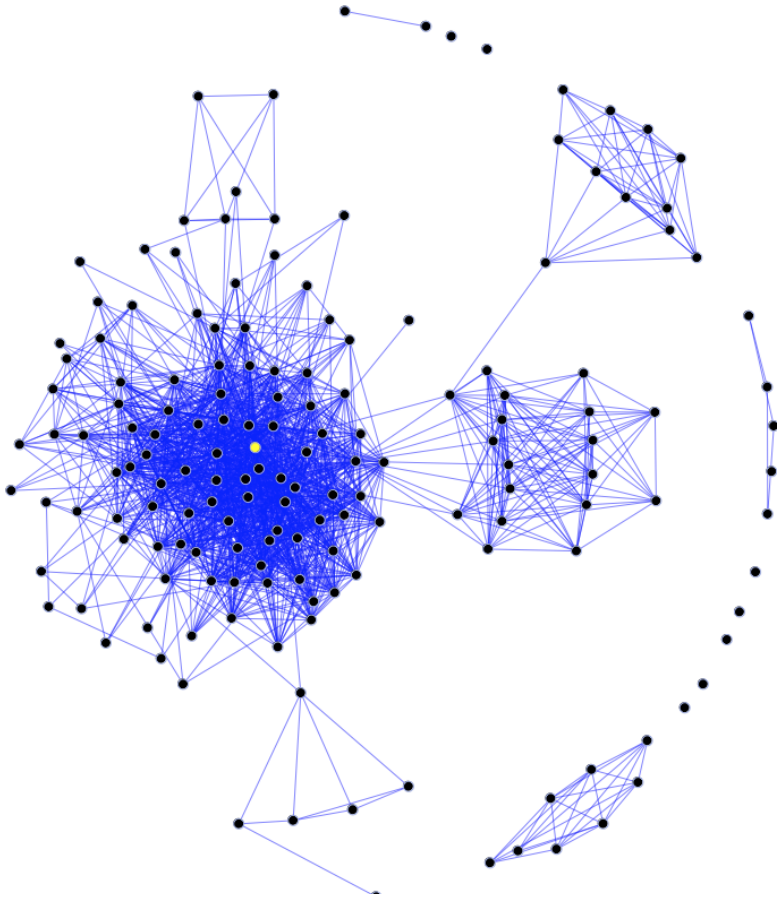
Big Data Problems

Linear algebra

Graph problems

As with other problems - big data graphs are like HPC graphs, but more so.

Very sparse, very irregular: nodes can have enormously varying degrees, *e.g.* social graphs



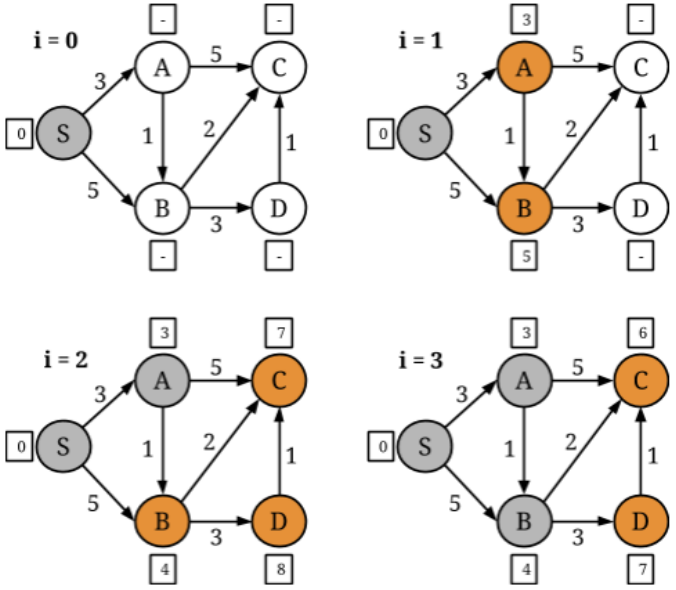
Big Data Problems

Linear algebra

Graph problems

Generally decomposed in similar ways.

Processing looks very much like neighbour exchange on an unstructured mesh; can map unstructured mesh computations onto (very regular) graph problems.



https://flink.apache.org/news/2015/08/24/introducing-flink-gelly.html

Big Data Problems

Linear algebra

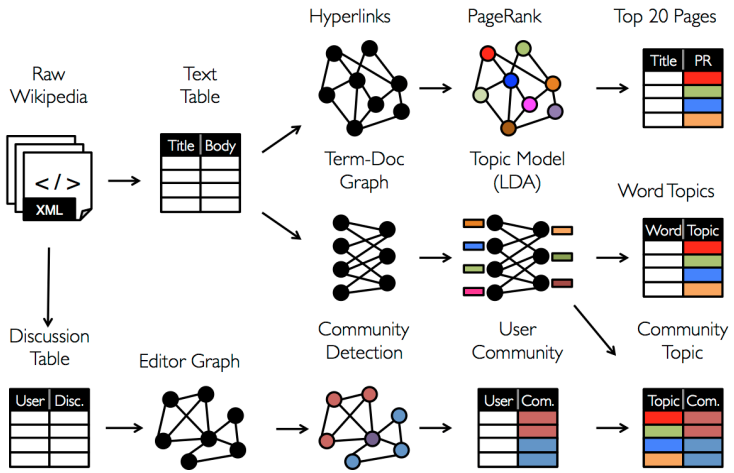
Graph problems

Calculations on (*e.g.*) social graphs are typically very low-compute intensity:

- Sum
- Min/Max/Mean

So that big-data graph computations are often *more* latency sensitive than more compute-intensive technical computations

⇒ lots of work done and in progresss to reduce communication/framework overhead



https://spark.apache.org/docs/1.2.1/graphx-programming-guide.html

Big Data Problems

Linear algebra

Graph problems

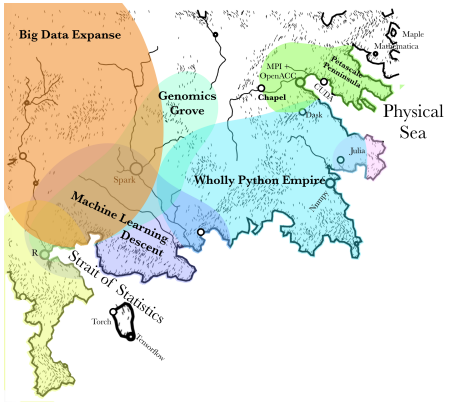
Commonalities

The problems big-data practitioners face are either:

- The same as in traditional HPC
- The same as new scientific computing fields
- Or what data analysis/HPC will be facing towards exascale
  - Less regular/structured
  - More dynamic

The Present Day, 2017

Will argue that they are not so different, and there's a lot to learn (on both sides) across the data science/simulation science divide



- Simulations are getting more complex, dynamic
- Big data problems have long been in-memory, increasingly compute intensive
- Moving towards each other in fits and starts

I tend to place Chapel as a redoubt on the outskirts of traditional HPC terrain, trying to lead the community towards where the action is:

- Productive tooling
- Modern language affordances
- Making it easier to tackle scale, more complex problems

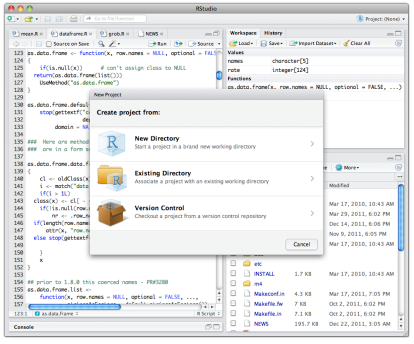
R: https://www.r-project.org

R

Overview

The R foundation considers R "an environment within which statistical techniques are implemented."

- A programming language built around statistical analysis and (primarily) interactive use.
- Enormous contributed package library CRAN (10,700+ packages).
- Lingua Franca of desktop statistical analysis.
- Lovely newish development/interactive use environment, RStudio.
- *Huge* in biostatistics: Bioconductor



R's popularity was *not* a given.

- Many extremely established incumbant stats packages, commercial (SPSS, SAS)
- Referees can always say "I don't trust this new program, what does good old SPSS/SAS say? (Fear may be more important than actual fact).
- Free, easily extensible, high-quality — took ages to catch on, but it did.

Lesson 1: Incumbents *can* be beaten.

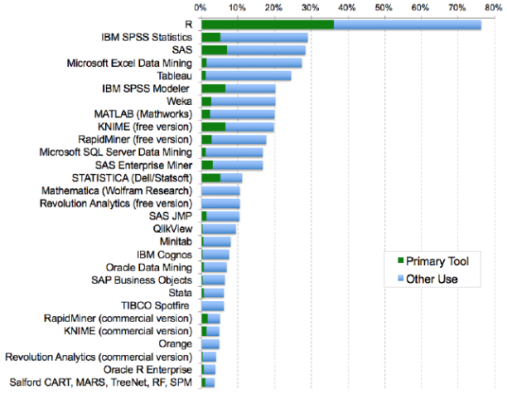


Figure from http://r4stats.com/articles/popularity/

R's popularity was *not* a given.

- Many extremely established incumbant stats packages, commercial (SPSS, SAS)
- Referees can always say "I don't trust this new program, what does good old SPSS/SAS say? (Fear may be more important than actual fact).
- Free, easily extensible, high-quality --- took ages to catch on, but it did.

Lesson 2: Growth is slow, until it isn't.

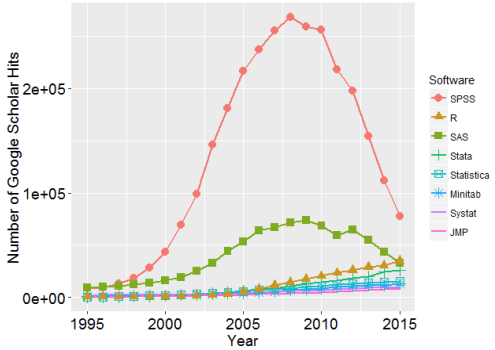


Figure from http://r4stats.com/articles/popularity/

R

Overview

Initial History

A big reason for deciding to use R are the packages that are available

- High-quality, user-contributed packages to solve specific types of problems
- Written to solve authors' problem, helpful to others

Lesson 3: Users' contributions can be as important for adoption as implementers'.

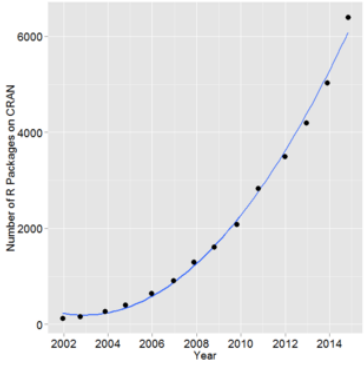


Figure from http://blog.revolutionanalytics.com/2016/04/cran-package-growth.html

The fundamental data structure of R has been(*) the `dataframe`.

- Think spreadsheet
- List of typed columns (1d vectors)
- Can be thought of as 1d array of `record`.

Name (character)	Age (numeric)	Favourite Language (factor)	Creature Type (factor)	Fully Charged? (logical)
"Jonathan"	NA	Chapel!	Person	FALSE
"Sammy"	2	R	Dog	TRUE
"Harvey"	5	Assembly	Roomba	TRUE
...	...	...	...	...

R

Overview

Initial History

Dataframes

The fundamental data structure of R has been(*) the `dataframe`.

- Think spreadsheet
- List of typed columns (1d vectors)
- Can be easily thought of as 1d array of `record`.
- Easily distributed over multiple machines!

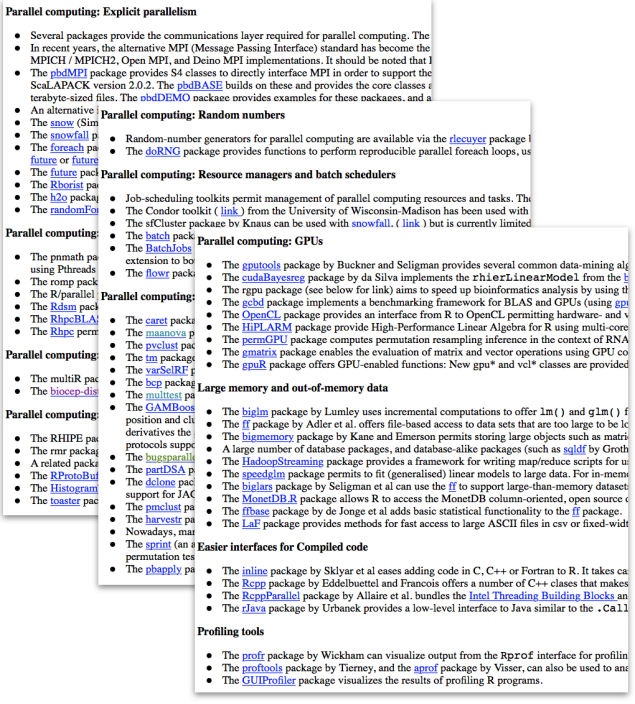
Name (character)	Age (numeric)	Favourite Language (factor)	Creature Type (factor)	Fully Charged? (logical)
"Jonathan"	NA	Chapel!	Person	FALSE
"Sammy"	2	R	Dog	TRUE

Name (character)	Age (numeric)	Favourite Language (factor)	Creature Type (factor)	Fully Charged? (logical)
"Harvey"	5	Assembly	Roomba	TRUE
...	...	...	...	...

The fundamental data structure of R has been the `dataframe`.

- Easily distributed over multiple machines!

One might reasonably expect that there thus would be a thriving ecosystem of parallel/big data tools for R. There's *some* truth to that (*e.g.* CRAN HPC Task view):



Parallel computing: Explicit parallelism

- Several packages provide the communications layer required for parallel computing. The
- In recent years, the alternative MPI (Message Passing Interface) standard has become the MPICH / MPICH2, Open MPI, and Deino MPI implementations. It should be noted that
- The pbdMPI package provides S4 classes to directly interface MPI in order to support the ScaLAPACK version 2.0.2. The pbdBASE builds on these and provides the core classes
- An alternative
- The snow (Sim
- The snowfall
- The foreach
- The future
- The Rhorist
- The h2o
- The randomFo

Parallel computing:

- The pnmath
- The romp
- The R/parallel
- The Rdsm
- The RhpcBLA
- The Rhpc

Parallel computing:

- The multiR
- The biocep-di

Parallel computing:

- The RHIPE
- The rmr
- A related
- The RProtoBu
- The Histogram
- The toaster

Parallel computing: Random numbers

- Random-number generators for parallel computing are available via the rlecuyer package
- The doRNG package provides functions to perform reproducible parallel foreach loops, us

Parallel computing: Resource managers and batch schedulers

- Job-scheduling toolkits permit management of parallel computing resources and tasks. Th
- The Condor toolkit (link) from the University of Wisconsin-Madison has been used with
- The sfCluster package by Knaus can be used with snowfall. (link) but is currently limited
- The batch
- The BatchJobs extension to bo
- The flowr

Parallel computing:

- The caret
- The maanova
- The pvclust
- The tm
- The varSelRF
- The bcp
- The multtest
- The GAMBoos position and cl derivatives the protocols supp
- The bugsparall
- The partDSA
- The dclone support for JA
- The pmclust
- The harvestr
- Nowadays, ma
- The sprint (an permutation te
- The pbapply

Parallel computing: GPUs

- The gputools package by Buckner and Seligman provides several common data-mining alg
- The cudaBayesreg package by da Silva implements the `rhierLinearModel` from the
- The rgpu package (see below for link) aims to speed up bioinformatics analysis by using t
- The gcbd package implements a benchmarking framework for BLAS and GPUs (using gp
- The OpenCL package provides an interface from R to OpenCL permitting hardware- and
- The HiPLARM package provide High-Performance Linear Algebra for R using multi-core
- The permGPU package computes permutation resampling inference in the context of RNA
- The gmatrix package enables the evaluation of matrix and vector operations using GPU co
- The gpuR package offers GPU-enabled functions: New gpu* and vcl* classes are provided

Large memory and out-of-memory data

- The biglm package by Lumley uses incremental computations to offer `lm()` and `glm()`
- The ff package by Adler et al. offers file-based access to data sets that are too large to be lo
- The bigmemory package by Kane and Emerson permits storing large objects such as matri
- A large number of database packages, and database-alike packages (such as sqldf by Groth
- The HadoopStreaming package provides a framework for writing map/reduce scripts for u
- The speedglm package permits to fit (generalised) linear models to large data. For in-mem
- The biglars package by Seligman et al can use the ff to support large-than-memory dataset
- The MonetDB.R package allows R to access the MonetDB column-oriented, open source
- The ffbase package by de Jonge et al adds basic statistical functionality to the ff package.
- The LaF package provides methods for fast access to large ASCII files in csv or fixed-widt

Easier interfaces for Compiled code

- The inline package by Sklyar et al eases adding code in C, C++ or Fortran to R. It takes ca
- The Rcpp package by Eddelbuettel and Francois offers a number of C++ clases that makes
- The RcppParallel package by Allaire et al. bundles the Intel Threading Building Blocks an
- The rJava package by Urbanek provides a low-level interface to Java similar to the `.Call`

Profiling tools

- The profr package by Wickham can visualize output from the `Rprof` interface for profilin
- The proftools package by Tierney, and the aprof package by Visser, can also be used to ana
- The GUIProfiler package visualizes the results of profiling R programs.

But a large number of packages isn't necessarily a sign of vibrancy

- Can be wheel reinvention factory

R has several (solid, well made) parallel packages: snow, multicore (now both in core), foreach.

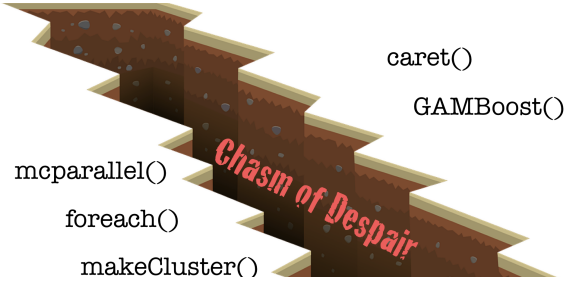
- But they don't work together
- And don't implement any higher-order algorithms.

Also has several excellent packages that make use of parallelism:

- Caret (various data mining algorithms)
- BiocParallel (for Bioconductor packages)

But these represent a *lot* of work by people; hard to get from one side to the other.

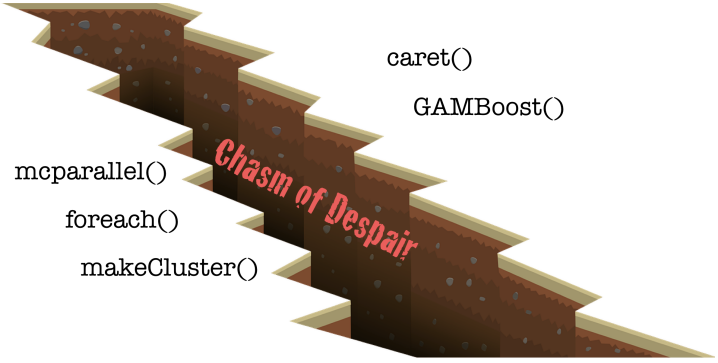
SparkR allows you to run R code through Spark, but impedence mismatch between paradigms.



If your parallelism isn't very easily expressed, and a higher-level package for solving your problems doesn't already exist, you have to parallelize your algorithms from very basic pieces

- But scientists don't want to write parallel code
- They just want to solve their problems!

Lesson 4: Decompositions aren't enough — need rich, composable, parallel tools.



Focused entirely on statistical computing (pro or con)

Cons

- Hit-or-miss support for parallel computations
- Purely interpreted; pure R is slow

Pros

- Widespread adoption
- Enormous package support (many written in C++)
- Close to dominant on the desktop (with Python/Pandas nipping at heels)

Spark: http://spark.apache.com

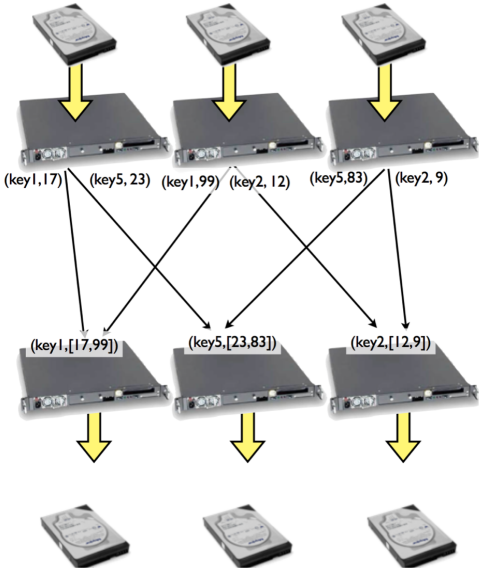
Spark

Overview

Hadoop came out in ~2006 with MapReduce as a computational engine, which wasn't that useful for scientific computation.

- One pass through data
- Going back to disk every iteration

However, the ecosystem flourished, particularly around the Hadoop file system (HDFS) and new databases and processing packages that grew up around it.



Spark

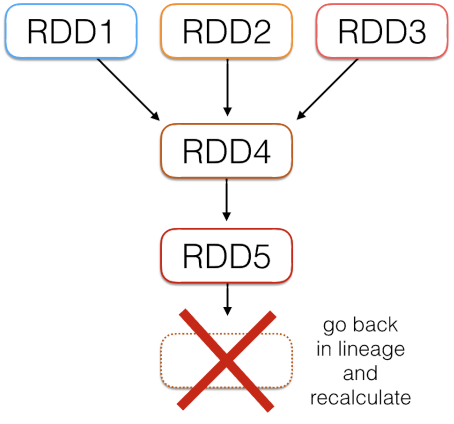
Overview

Spark (2012) is in some ways "post-Hadoop"; it can happily interact with the Hadoop stack but doesn't require it.

Built around concept of in-memory resilient distributed datasets

- Tables of rows, distributed across the job, normally in-memory
- Immutable
- Restricted to certain transformations

Used for database, machine learning (linear algebra, graph, tree methods), *etc.*



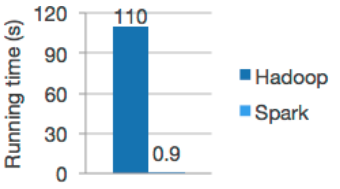
Spark

Overview

Performance

Being in-memory was a huge performance win over Hadoop MapReduce for multiple passes through data.

Spark immediately began supplanting MapReduce for complex calculations.



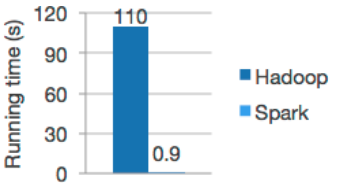
Lesson 6: Performance is crucial!

Spark

Overview

Performance

Being in-memory was a huge performance win over Hadoop MapReduce for multiple passes through data.



Spark immediately began supplanting MapReduce for complex calculations.

Lesson 6: Performance is crucial!

...To a point.

In 2012, either would have been much faster in MPI or a number of HPC frameworks.

- No multicore
- Generic sockets for communications
- No GPUs
- JVM: Garbage collection jitter, pausses

But development time, lack of fault tolerance, no integration into ecosystem (HDFS, HBase..) mean that not even considered.

Don't have to be faster than *everything*.

Spark

Overview

Performance

Project Tungsten (2015) was an extensive rewriting of core Spark for performance.

- Get rid of JVM memory management, handle it themselves (FORTRAN77 workspace arrays!)
- Vastly improved cache performance
- Code generation (more later)

In 2016, built-in GPU support.

Lesson 8: There will *always* be pending performance improvements. They're important, but not show-stoppers.

Spark

Overview

Performance

Project Tungsten (2015) was an extensive rewriting of core Spark for performance.

- Get rid of JVM memory management, handle it themselves (FORTRAN77 workspace arrays!)
- Vastly improved cache performance
- Code generation (more later)

In 2016, built-in GPU support.

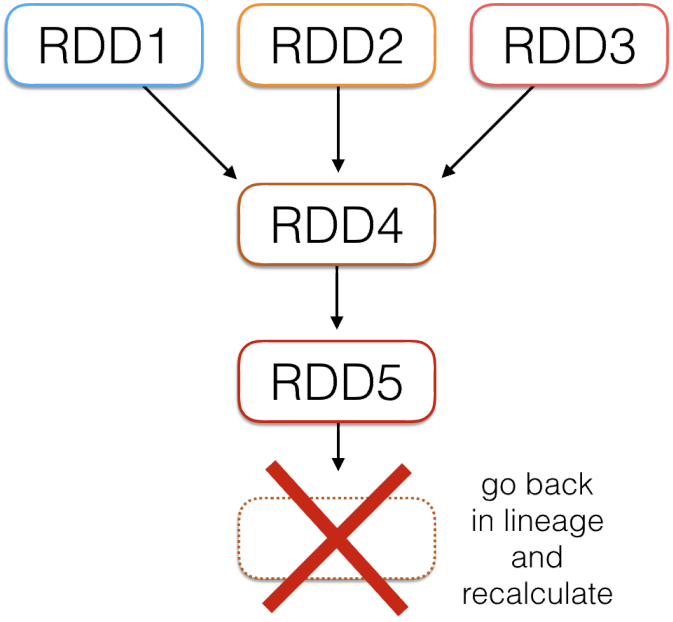
Lesson 8: There will *always* be pending performance improvements. They're important, but not show-stoppers.

Lesson 9: Big Data frameworks are learning HPC lessons faster than HPC stacks are learning Big Data lessons.

Operations on Spark RDDs can be:

- Transformations, like map, filter, reduce, join, groupBy...
- Actions like collect, foreach, ..

You build a Spark computation by chaining together transformations; but no data starts moving until part of the computation is materialized with an action.



Spark

Overview

Performance

RDDs

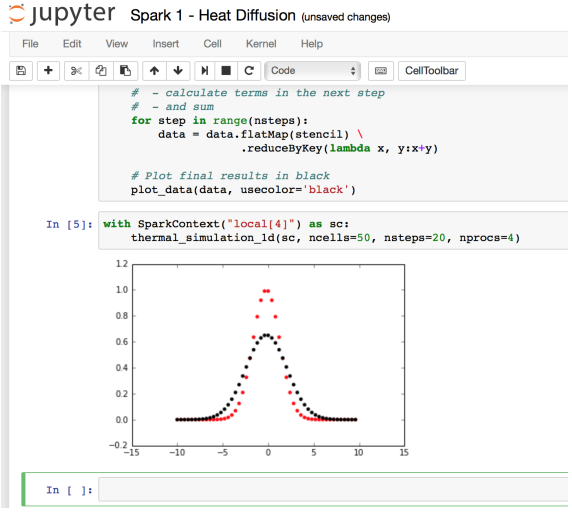
Spark RDDs prove to be a very powerful abstraction.

Key-Value RDDs are a special case - a pair of values, first is key, second is value associated with.

Linda tuple spaces, which underly Gaussian.

Can easily use join, *etc.* to bring all values associated with a key together:

- Like all stencil terms that are contribute at a particular grid point



But RDDs are also building blocks.

Spark Dataframes are lists of columns, like pandas or R data frames.

Can use SQL-like queries to perform calculations. But this allows bringing the entire mature machinery of SQL query optimizers to bear, allowing further automated optimization of data movement, and computation.

(Spark Notebook 2)

jupyter Spark 2 - Data Frames Last Checkpoint: 3 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

Code CellToolbar

In [15]:
```
df = df.select("id",
                  rand(seed=10).alias("uniform"),
                  randn(seed=27).alias("normal"))
df.show(10)
```

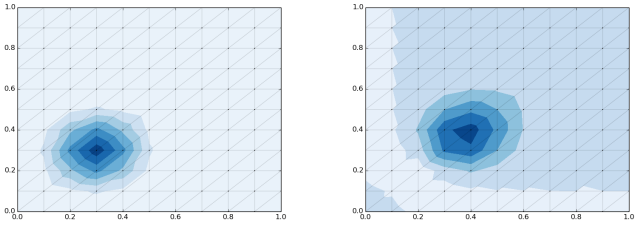
```
+---+-------------------+-------------------+
| id|            uniform|             normal|
+---+-------------------+-------------------+
|  0|0.41371264720975787| 0.5888539012978773|
|  1| 0.7311719281896606| 0.8645537008427937|
|  2| 0.9031701155118229| 1.2524569684217643|
|  3|0.09430205113458567| -2.573636861034734|
|  4|0.38340505276222947| 0.5469737451926588|
|  5| 0.5569246135523511|0.17431283601478723|
|  6| 0.4977441406613893|-0.7040284633147095|
|  7| 0.2076666106201438| 0.4637547571868822|
|  8| 0.9571919406508957|  0.920722532496133|
|  9| 0.7429395461204413|-1.4353459012380192|
+---+-------------------+-------------------+
only showing top 10 rows
```

Graph library — GraphX — has also been implemented on top of RDDs.

Many interesting features, but for us: Pregel-like algorithms on graphs.

This makes implementing unstructured mesh methods extremely straightforward (Spark notebook 4):

```
def step(g:Graph[nodetype, edgetype]) : Graph[nodetype, edgety
    val terms = g.aggregateMessages[msgtype](
        // Map
        triplet => {
            triplet.sendToSrc(src_msg(triplet.attr, triplet.sr
            triplet.sendToDst(dest_msg(triplet.attr, triplet.s
          },
        // Reduce
        (a, b) => (a._1, a._2, a._3 + b._3, a._4 + b._4, a._5
    )

    val new_nodes = terms.mapValues((id, attr) => apply_update

    return Graph(new_nodes, graph.edges)
}
```



All of these features - key-value RDDs, Dataframes, (now Datasets), and graphs, are built upon the basic RDD plus the fundamental transformations.

Lesson 4b: The right abstractions — decompositions with enough primitive operations to act on them — can be enough to build an ecosystem on

Delayed computation + view of entire algorithm allows optimizations over the entire computation graph.

So for instance here, nothing starts happening in earnest until the `plot_data()` (Spark notebook 1)

```
# Main loop: For each iteration,
#  - calculate terms in the next step
#  - and sum
for step in range(nsteps):
    data = data.flatMap(stencil) \
                .reduceByKey(lambda x, y:x+y)

# Plot final results in black
plot_data(data, usecolor='black')
```

Knowledge of lineage of every shard of data also means recomputation is straightforward in case of node failure

Spark

Overview

Performance

RDDs

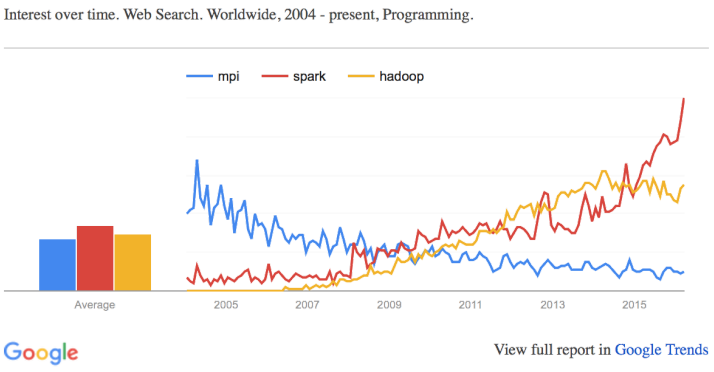
Dataframes

Graphs

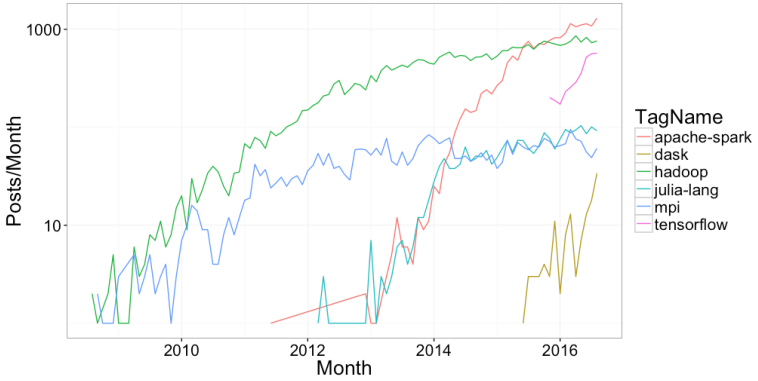
Execution graphs

Adoption in Science

Adoption has been enormous *broadly*:



Google Search



Questions on Stack Overflow

But comparatively little uptake in science yet - even though it seems like it would be right at home in large-scale genomics:

- Graph problems
- Large statistical analyses

(GATK is a bit of a special case - more research infrastructure than a research tool per se)

But comparatively little uptake in science yet - even though it seems like it would be right at home in large-scale genomics:

- Graph problems
- Large statistical analyses

(GATK is a bit of a special case - more research infrastructure than a research tool per se)

My claim is that its heavyweight nature is an awkward fit for scientist patterns of work

- Noodle around on laptop
- Develop methods, gain confidence on smaller data sets
- Scale up over time

Who spends months developing a method, tries it for the first time on 100TB of data, only to discover the approach is doomed to failure?

Lesson 10: For science, scale *down* may be as important as scale up

Cons

- JVM Based (Scala) means C interoperability always fraught.
- Not much support for high-performance interconnects (although that's coming from third parties - HiBD group at OSU)
- Very little explicit support for multicore yet, which leaves much performance on the ground.
- Doesn't scale *down* very well; very heavyweight

Pros

- Very rapidly growing
- Performance improvements version to version
- Easy to find people willing to learn

Dask: http://dask.pydata.org/

Dask

Overview

Dask is a python parallel computing package

- Very new - 2015
- As small as possible
- Scales down very nicely
- Adoption extremely fast

Dask

Overview

Dask is a python parallel computing package

- Very new - 2015
- As small as possible
- Scales down very nicely
- Adoption extremely fast

- Works very nicely with NumPy, Pandas, Scikit-Learn
- Is definitely nibbling into HPC “market share”
  - For traditional numerical computing on few nodes
  - For less regular data analysis/machine learning on larger scale
  - (likely siphoning off a little uptake of Spark, too)

Used for very general data analysis (linear algebra, trees, tables, stats, graphs...) and machine learning

Lesson 11: Library support vital

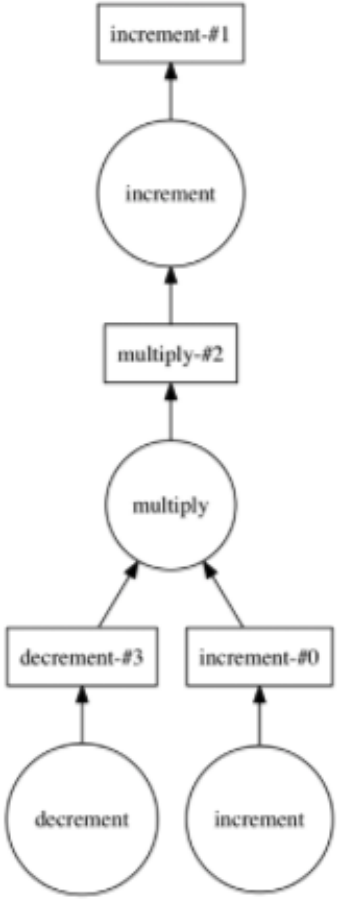
Dask

Overview

Task Graphs

Allows manual creation of quite general parallel computing data flows (making it a great way to prototype parallel numerical algorithms):

```
from dask import delayed, value

@delayed
def increment(x, inc=1):
    return x + inc

@delayed
def decrement(x, dec=1):
    return x - dec

@delayed
def multiply(x, factor):
    return x*factor

w = increment(1)
x = decrement(5)
y = multiply(w, x)
z = increment(y, 3)

from dask.dot import dot_graph
dot_graph(z.dask)

z.compute()
```



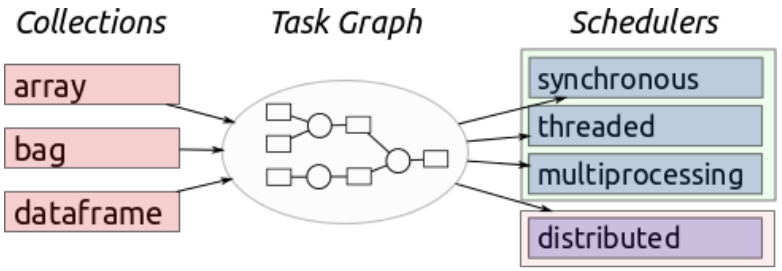
Dask

Overview

Task Graphs

Once the graph is constructed, computing means scheduling either across threads, processes, or nodes

- Redundant tasks (recomputation) pruned
- Intermediate tasks discarded after use
- Memory use kept low
- If guesses wrong, task dies, scheduler retries
    - Fault tolerance



http://dask.pydata.org/en/latest/index.html

Dask

Overview

Task Graphs

Dask Arrays

Array support also includes a small but growing number of linear algebra routines

Dask allows out-of-core computation on arrays (or dataframes, or bags of objects): will be increasingly important in NVM era

- Graph scheduler automatically pulls only chunks necessary for any task into memory
- New: intermediate results can be spilled to disk

```
file = h5py.File(hdf_filename,'r')
mtx = da.from_array(file['/M'], chunks=(1000, 1000))
u, s, v = da.linalg.svd(mtx)
u.compute()
```

Lesson 12: With NVMe, out-of-core is coming back, and some packages are already thinking about it

Dask

Overview

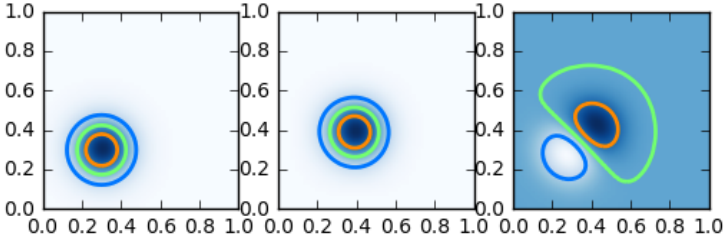
Task Graphs

Dask Arrays

Arrays have support for guardcells, which make certain sorts of calculations trivial to parallelize (but lots of copying right now):

(From dask notebook)

```
subdomain_init = da.from_array(dens_init, chunks=((npts+1)//2

def dask_step(subdomain, nguard=2):
    # `advect` is just operator on a numpy array
    return subdomain.map_overlap(advect, depth=nguard, bounda

with ResourceProfiler(0.5) as rprof, Profiler() as prof:
    subdomain = subdomain_init
    nsteps = 100
    for step in range(0, nsteps//2):
        subdomain = dask_step(subdomain)
    subdomain = subdomain.compute(num_workers=2, get=mp_get)
```



Dask

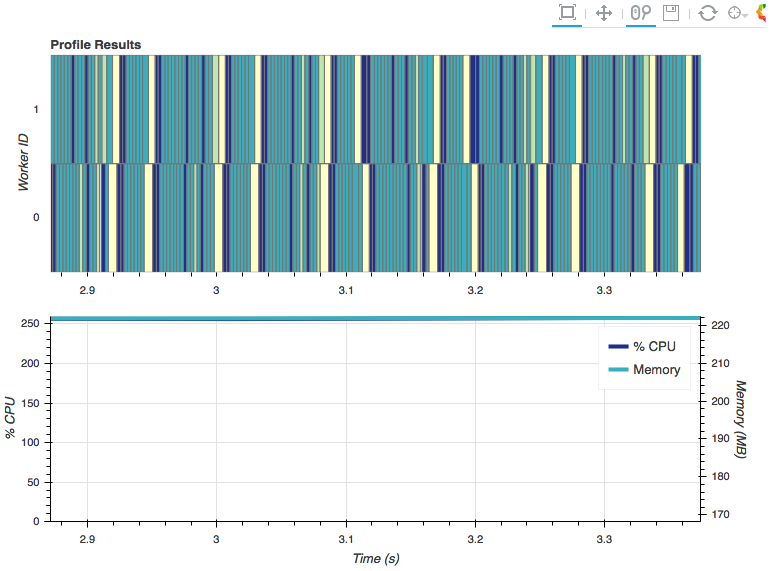
Overview

Task Graphs

Dask Arrays

Diagnostics

Comes with several very useful performance profiling tools which will be instantly famiilar to HPC community members



Dask

Overview

Task Graphs

Dask Arrays

Diagnostics

Pros/Cons

Not going to be a killer platform for solving PDEs just yet

- I claim this is because you can't hint strongly enough to scheduler yet about data placement

Could easily be of interest in very near term for large-scale biostatistical data analysis (scikit-learn).

Out-of-core analysis makes scale down even more interesting.

Nothing really there for graph problems, but it's not impossible in the medium term.

Cons

- Performance: Aimed at analysis tasks (big, more loosely coupled) rather than simulation
  - Scheduler+TCP: 200 μs per-task overhead, orders of magnitude larger than an MPI message
  - Single scheduler processes
  - Not intended as replacement in general for large-scale tightly-coupled computing

Pros

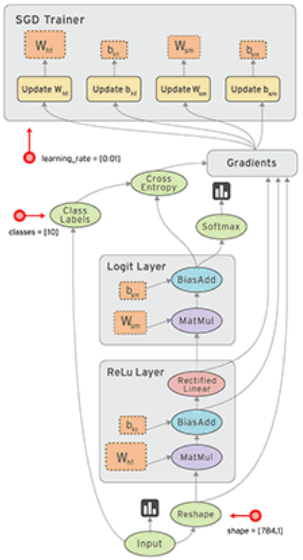
- Trivial to install, start using
- Outstanding for prototyping parallel algorithms
- Out-of-core support baked in
- With Numba+Numpy, reasonable single-core performance (~factor of 2 of Chapel)
- Automatically overlaps communication with computation: 200 μs might not be so bad for some methods
- Scheduler, communications all in pure python right now, rapidly evolving:
  - **Much** scope for speedup

TensorFlow: http://tensorflow.org

TensorFlow

Overview

TensorFlow is an open-source dataflow for numerical computation with dataflow graphs, where the data is always in the form of “tensors” (n-d arrays).



Very new: Released November 2015

From Google, who uses it for deep learning and othe rmachine learning tasks.

Lots of BLAS operations and function evaluations but also general numpy-type operations, can use GPUs or CPUs.

Deep learning: largely (but not exclusively) about breaking data (training set) into large chunks, performing calculations, and updating each other with updates from those calculations synchronously or asynchronously.

Lesson 13: Parts of “big data” are getting very close to traditional HPC problems.

TensorFlow

Overview

Graphs

As an example of how a computation is set up, here is a linear regression example.

TensorFlow notebook 1

In [11]:
```
# Try to find values for W and b that compute y_data = W * x_data + b
# (We know that W should be 0.1 and b 0.3, but Tensorflow will
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# Before starting, initialize the variables.  We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in range(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))
```

TensorFlow

Overview

Graphs

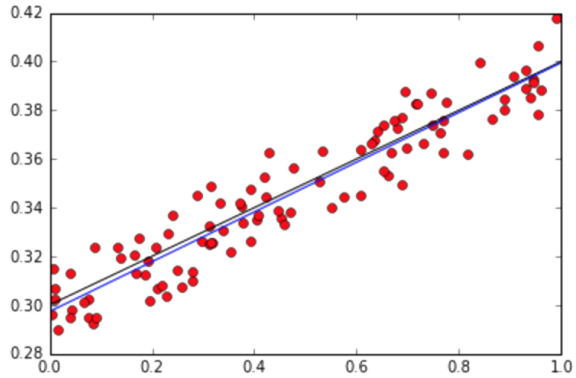
Linear regression is already built in, and doesn't need to be iterative, but this example is quite general and shows how it works.

Variables are explicitly introduced to the TensorFlow runtime, and a series of transformations on the variables are defined.

When the entire flowgraph is set up, the system can be run.

The integration of tensorflow tensors and numpy arrays is very nice.

```
Out[12]: [<matplotlib.lines.Line2D at 0x7f9fc1d3c358>]
```



TensorFlow

Overview

Graphs

Mandelbrot

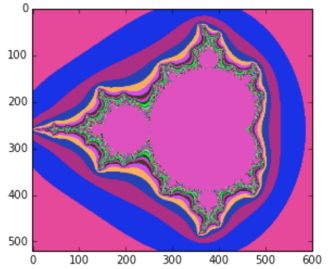
All sorts of computations on regular arrays can be performed.

Some computations can be split across GPUs, or (eventually) even nodes.

All are multi-threaded.

```
In [15]: # Compute the new values of z: z^2 + x
         zs_ = zs*zs + xs

         # Have we diverged with this new value?
         not_diverged = tf.complex_abs(zs_) < 4

         # Operation to update the zs and the iteration count.
         #
         # Note: We keep computing zs after they diverge! This
         #       is very wasteful! There are better, if a little
         #       less simple, ways to do this.
         #
         step = tf.group(zs.assign(zs_),
                         n_iters.assign_add(tf.cast(not_diverged, tf.float32)))

         for i in range(200):
             step.run()
```

```
In [16]: display_fractal(n_iters.eval())
```

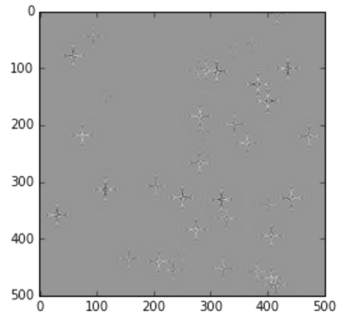


All sorts of computations on regular arrays can be performed.

Some computations can be split across GPUs, or (eventually) even nodes.

All are multi-threaded.

In [87]:
```
# Initialize state to initial conditions
tf.initialize_all_variables().run()

# Run 1000 steps of PDE
for i in range(1000):
  # Step simulation
  step.run({eps: 0.03, damping: 0.04})
  # Visualize every 50 steps
  if i % 50 == 0:
    display_array(U.eval())
```



As with laying out the computations, distributing the computations is still quite manual:

```
with tf.device("/job:ps/task:0"):
  weights_1 = tf.Variable(...)
  biases_1 = tf.Variable(...)

with tf.device("/job:ps/task:1"):
  weights_2 = tf.Variable(...)
  biases_2 = tf.Variable(...)

with tf.device("/job:worker/task:7"):
  input, labels = ...
  layer_1 = tf.nn.relu(tf.matmul(input, weights_1) + biases_1
  logits = tf.nn.relu(tf.matmul(layer_1, weights_2) + biases_
  # ...
  train_op = ...

with tf.Session("grpc://worker7.example.com:2222") as sess:
  for _ in range(10000):
    sess.run(train_op)
```

Communications is done using gRPC, a high-performance RPC library based on what Google uses internally.

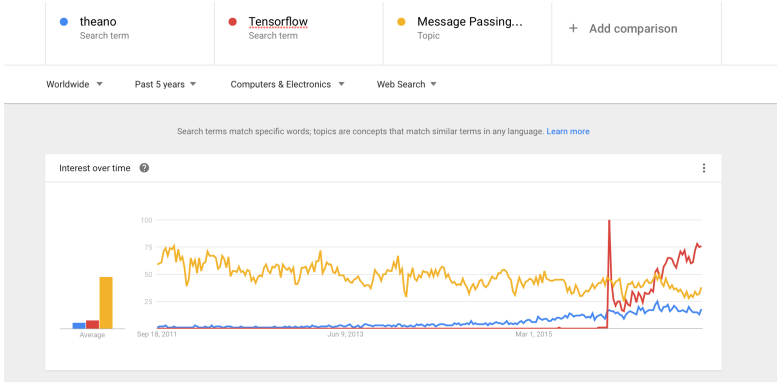
TensorFlow

Very rapid adoption, even though targetted very narrowly: deep learning

Overview

All threaded number crunching on arrays and communication of results of those array calculations

Graphs



Mandelbrot

Wave Eqn

Distributed

Adoption

Cons

- N-d arrays only means limited support for, e.g., unstructured meshes, hash tables (bioinformatics)
- Distribution of work remains limited and manual

Pros

- C++ - interfacing is much simpler than Spark
- Fast
- GPU, CPU support, not unreasonable to expect Phi support shortly
- Can make use of infrastructure for synchronous, asynchronous updates between data-parallel tasks
- Great for data processing, image processing, or computations on n-d arrays

Common Themes

Higher-Level Abstractions

- Spark: Resilient distributed data set (table), upon which:
  - Graphs
  - Dataframes/Datasets
  - Machine learning algorithms (which require linear algebra)
  - Mark of a good abstraction is you can build lots atop it!
- Dask:
  - Task Graph
  - Dataframe, array, bag operations
- TensorFlow:
  - Data flow
  - Certain kinds of “Tensor” operations

Common Themes

Higher-Level Abstractions

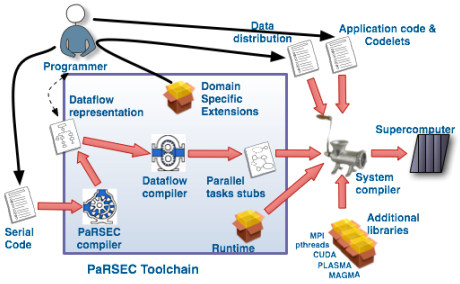
Data Flow

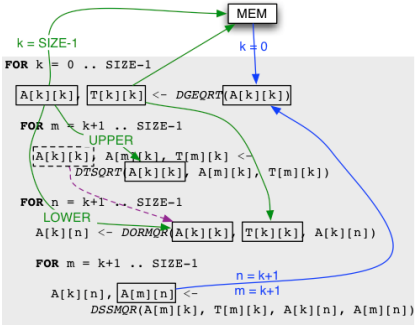
All of the approaches we've seen implicitly or explicitly constructed dataflow graphs to describe where data needs to move.

Then can build optimization on top of that to improve data flow, movement

These approaches are extremely promising, and already completely usable at scale for some sorts of tasks.

Already starting to attract attention in HPC, e.g. PaRSEC at ICL:





Julia: http://julialang.org

Julia

Overview

julia is "a high-level, high-performance dynamic programming language for numerical computing."

Like Chapel, aims to be productive, performant, parallel. Targets itself as a matlab-killer.

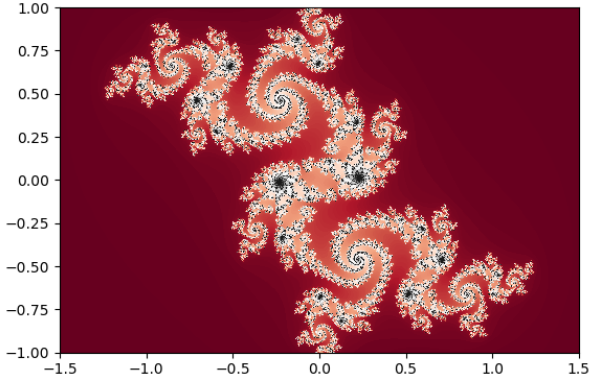
Most notable features:

- Dynamic language: JIT, rich types, multiple dispatch
  - Give a "scripting language" feel while giving performance closer to C or Fortran
- Lisp-like metaprogramming: Code is Data
  - With JIT, makes it possible to re-write Julia code on the fly
  - Makes it possible to write mini-DSLs for particular problem types: differential equations, optimization
- Full suite of parallel primitives

Julia

Overview

```
using PyPlot

# julia set
function julia(z, c; maxiter=200)
    for n = 1:maxiter
        if abs2(z) > 4
            return n-1
        end
        z = z*z + c
    end
    return maxiter
end

jset = [ UInt8(julia(complex(r,i), complex(-.06,.67)))
             for i=1:-.002:-1, r=-1.5:.002:1.5 ];
get_cmap("RdGy")
imshow(jset, cmap="RdGy", extent=[-1.5,1.5,-1,1]
```



Julia

Overview

Single-Core Performance

Single-core performance is very good, particularly for a JIT.

Test below is for a simple 1-d stencil calculation (https://www.dursi.ca/post/julia-vs-chapel.html)

time	Julia	Chapel	Numpy + Numba	Numpy
run	0.0084	0.0098 s	0.017 s	0.069 s
compile	0.57 s	4.8s	0.73 s	-

Julia edges out Chapel... but for this test, look at Python with Numpy and numba, only a factor of two behind.

Single-core performance has been the main focus of Julia, to the exclusion of almost all else - multithreading is still considered experimental.

Julia

Overview

Single-Core Performance

Distributed Data

Julia has a DistributedArray module, but it has very large overhead; better suited for merging data once at the end of long purely local computation (processing and then stacking images, etc)

Below is a test for running on 8 cores of a (single) node:

Julia		Chapel		Dask
-p=1	-p=8	-nl=1 tasks=8	-nl=8 tasks=1	workers=8
177s s	264 s	**0.4 s**	145 s	193 s

Lesson 14: Hierarchical approach to parallelism matters.

Need to be able to easily exploit threading, NUMA locality, cross-node communications...

Julia has good libraries for data analysis, modest support for graph algorithms, but all single-node; very little support for distributed memory computing.

Julia

Overview

Single-Core Performance

Distributed Data

Pros/Cons

Cons

- Very little performant support for distributed memory computing, not clear it is forthcoming

Pros

- Single core fast, and on-node fairly fast
- Very nice interactive use, works with Jupyter or REPL
- Some excellent libraries
- Very powerful platform for writing DSLs

My Benchmark Problems

My Benchmark Problems

So where does this leave my “curated” (read: wildly biased) set of benchmark problems?

In a dystopic world without efforts like Chapel, what would I be using?

My Benchmark Problems

PDEs

Heavy reliance on execution-graph optimizers has a lot of promise for highly dynamic simulations.

But where we are now, big Data frameworks aren't going to come save me from the current state of the art in large-scale PDE frameworks:

- Trilinos
- BoxLib
- ...

Amazing efforts, great tools, and the world is much better with them that it would be without them.

But huge code bases, very challenging to start with as a user, very difficult to make significant changes.

Based on MPI, which you may have heard I have opinions about.

My Benchmark Problems

PDEs

Genomics

Large genomics today means buying or renting very large (up to 1TB) RAM machines.

I'm starting to think that this reflects a failure of our parallel programming community.

Good news: there's lots of great work algorithmic being done in the genomics community

- Succinct data structures
- Approximate streaming methods

But this is work done because of scarsity, and the size of projects being tackled is being limited.

My Benchmark Problems

PDEs

Genomics

There are projects like HipMer (large-scale assembler, UPC++), but not a general solution.

GraphX for Spark could be useful, but only becomes performant on huge problems

- "Missing Middle" for where most of the work is, and for adoption

My Benchmark Problems

PDEs

Genomics

Biostatistics

Biostatistics is in exactly the same boat.

R works really, really well for ~desktop-scale problems.

Spark (or a number of other things) work if the data size starts large enough.

- Big international genomics projects

Death valley in between.

My Benchmark Problems

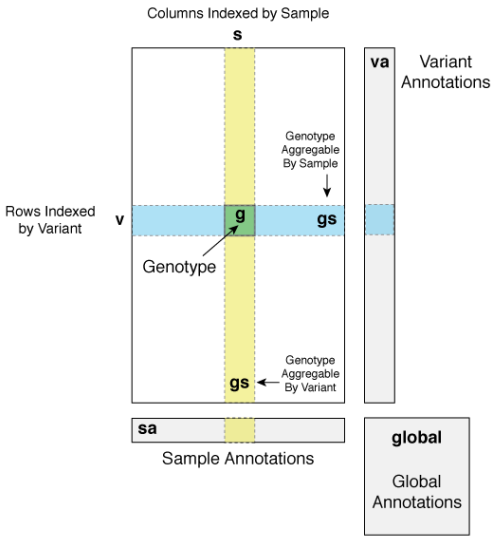
PDEs

Genomics

Biostatistics

Here's where we are now - the Broad institute in Boston put together the Hail project:

- Based on Spark
- "does person X have genetic variant Y" matrix of records
- Interactively query reductions of rows and columns
- A big problem is several billion entries. Future proof, but...
- This is not a hard problem!
- Very unwieldly for individual researchers on smaller sets.



Chapel

Chapel

So what does this mean for Chapel? Where does it sit in this landscape?

Chapel

So what does this mean for Chapel? Where does it sit in this landscape?

Here's my opinion, after casting about for langauges and frameworks for these sorts of problems:

- Chapel is *important*.
- Chapel is *mature*.
- Chapel is *just getting started*.

Chapel is...

Important

If the science community is going to have scientific frameworks designed *for our problems*, and not bolted on to LinkGoogBook's next big data framework, it's going to come from a project like Chapel.

Chapel is...

Important

If the science community is going to have scientific frameworks designed *for our problems*, and not bolted on to LinkGoogBook's next big data framework, it's going to come from a project like Chapel.

Using MPI as a framework just isn't sustainable for increasingly complex problems.

Chapel is...

Important

If the science community is going to have scientific frameworks designed *for our problems*, and not bolted on to LinkGoogBook's next big data framework, it's going to come from a project like Chapel.

Using MPI as a framework just isn't sustainable for increasingly complex problems.

Big data frameworks don't have any incentive to support scale-down, or tightly-coupled computing.

Chapel is...

Important

If the science community is going to have scientific frameworks designed *for our problems*, and not bolted on to LinkGoogBook's next big data framework, it's going to come from a project like Chapel.

Using MPI as a framework just isn't sustainable for increasingly complex problems.

Big data frameworks don't have any incentive to support scale-down, or tightly-coupled computing.

Scientists need both.

Chapel is...

$\{x | x \in \text{Projects Like Chapel}\} \equiv \text{Chapel}$

Important

Mature

Chapel is...

Important

Mature

$$\{x | x \in \text{Projects Like Chapel}\} \equiv \text{Chapel}$$

There are other research projects in this area - productive, performant, parallel computing languages for distributed-memory scientific computing.

But Chapel, especially now with 1.15, is a mature product.

Chapel is...

Important

Mature

$$\{x | x \in \text{Projects Like Chapel}\} \equiv \text{Chapel}$$

There are other research projects in this area - productive, performant, parallel computing languages for distributed-memory scientific computing.

But Chapel, especially now with 1.15, is a mature product.

It is crossing the barrier of “Fast Enough” for the problems that map naturally to it.

It has the pieces to start expanding that set of problems.

Chapel

Important

Mature

Just Getting Started

Has a very solid base.

- Native compilation, non-crazy runtime: scales down well.
- Good single core performance.
- Strong distributed-memory performance for rectangular dense or sparse arrays.
- Excellent set of parallel primitives.
- Useful tools.

Chapel

Important

Mature

Just Getting Started

I claim that there's enough of a foundation to start building an ecosystem around.

- *e.g.*, in or close to the Spark regime, not the R regime

But may still have to help users across their own “Crevace of Discouragement”

- Make it so easy for a scientist to start using Chapel for their problems it's too hard to resist.

Existing HPC stack helps with this!

- Many excellent existing tools
- That are incredibly difficult to start using

User community can contribute significantly to this.

Chapel

Important

Mature

Just Getting Started

Large Linear Solves?

PETSc is a widely used library for large sparse iterative solves.

- Excellent and comprehensive library of solvers
- It is the basis of a significant number of home-made simulation codes
- It is notoriously hard to start getting running with; nontrivial even for experts to install.

Significant fraction of PETSc functionality is tied up in large CSR matrices of reasonable structure partitioned by row, vectors, and solvers built on top.

What would a Chapel API to PETSc look like?

What would a Chapel implementation of some core PETSc solvers look like?

How about Scalapack?

Chapel

Important

Mature

Just Getting Started

Large Linear Solves?

Genomics?

Graph and string problems in genomics is:

- Huge: vastly larger than Astrophysics, which is where I come from
- Badly underserved
- Competition is threaded or even serial code on a single big memory machine
  - *e.g.*, lots of very nice code using Python dicts
  - And no numba or numpy equivalent to speed up these sorts of operations

Chapel already has associative, unstructured domains - what do some simple genomics tasks look like in Chapel?

Chapel

Important

Mature

Just Getting Started

Large Linear Solves?

Genomics?

Data Science?

Still this missing middle problem:

- Nothing (yet) can span the range of both R and Spark
- Python is making inroads
- Parts of the pieces are there:
  - partitioned arrays of records
- But would need other things
  - shuffles, very dynamic resizing
  - adoption may depend too strongly on libraries; R interop?

Chapel

Important

Mature

Just Getting Started

Large Linear Solves?

Genomics?

Data Science?

Glorious Age Of Expansion

Chapel has established a stronghold on the outskirts of modestly hostile territory.

But there are scientists in neighboring territories who need help.

In almost any direction, there are communities that would love what Chapel offers;

- Productivity
- Performance
- Desktop-to-Cluster scalability.

Future's wide open!