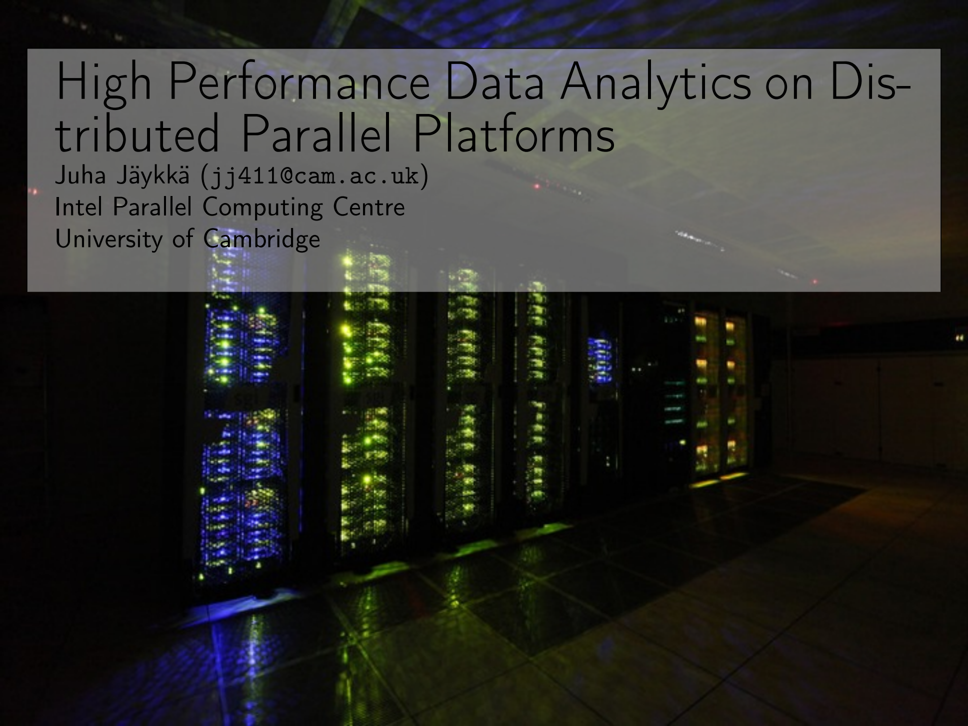
High Performance Data Analytics on Distributed Parallel Platforms

Juha Jäykkä (jj411@cam.ac.uk)
Intel Parallel Computing Centre
University of Cambridge



HPC clusters and programming

History

- mainly evolved out of two needs
  - access to more memory than can be harnessed with a single system
  - bear more compute power to bear on a fixed size problem
- the dominant problems: linear algebra and PDEs
  - (dense) linear algebra is highly structured
  - PDE operate on data using *stencils*
  - PDEs can be highly structured or somewhat less so, even unstructured

Hardware design driven by the above drivers

- also simplest to accommodate on hardware level
  - not everything maps well, but structured data is predictable so allows many helpful tricks
- one could say HPC arch is designed for data distributed programming
  - but no one has yet figured out how to efficiently scale up non-structured data

Why is structured data good?

Physics of Computer Memory

- CPU will fetch a cache line at a time
  - only really helps structured data
  - useless for random access
  - prefetcher
    - you will hear about them later
    - help only sequential access
- access is actually parallel typically 8 bytes at a time
  - in practice these are sequential memory locations
- linear, sequential access fastest

Structured vs Unstructured Access

- speed difference 1-2 orders of magnitude
- unfortunately unstructured is often unavoidable
- actually "unstructured" is a misnomer, all it means is complicated

What can we do? What do we want?

"higher frequency cpu"

- higher clocks ⇒ higher voltages ⇒ higher energy consumption
- silicon only conducts heat at a particular rate ⇒ melt-down

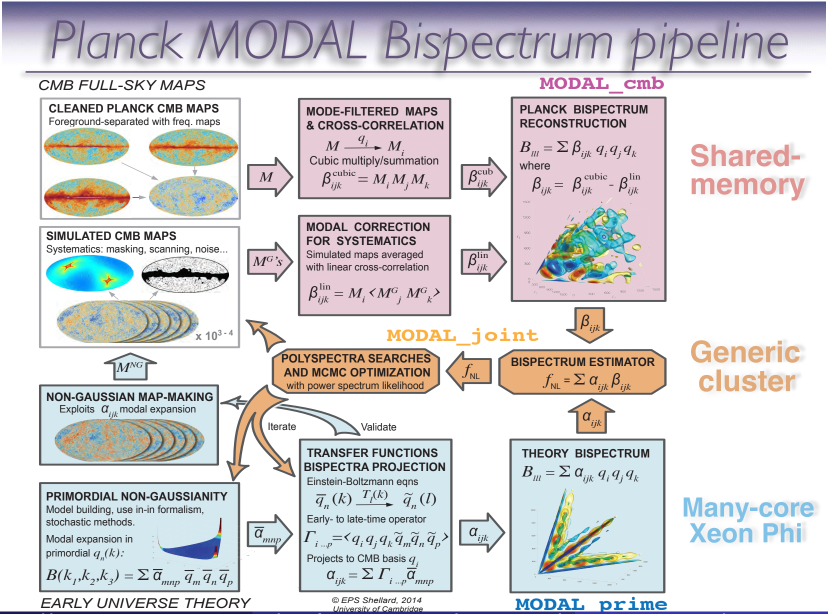
"more memory bandwidth"

- higher clocks again
- could do wider memory channels but £££
  - and would not help non-sequential access much anyway

"less latency"

- this is easy — downside is the bandwidth goes down
- oh, you want both? ok, but £££ £££ and £££ using very wide channels and old tech
- increasing Hz is near physics limit: light only tavels 30 cm in 1 ns

complex workflow pipeline for analysing Planck CMB data



MODAL pipeline

Images of the Sky (in microwaves)

- structured data, but with an unfriendly structure:
- a clever encoding of the celestial sphere where each pixel covers same area but
  - needs a *lookup array* to access actual data array: `imagedata[pixeladdress[x]]`
  - such indirect access is potentially bad for performance

Processing the images

- rather traditional numerical code (e.g. no strings to process)
- lots of integrals
- compute cross correlations

GW150914 (about 30 TB raw data)

GW150914

Gravitational Waves and other GR

Traditional: solve PDEs

- data sits in a regular, structured mesh
- data operated on with a stencil (e.g. discrete Laplacian)
- for GW150914 simulating the final "chirp"
  - with full GR would take about $O(100)$ PB
  - most cases can be dealt with post-Newtonian approximation

Modern codes use Adaptive Mesh Refinement

- regular structured data *within* regular structured data
  - data chopped into small chunks $\Rightarrow$ performance issues
  - still benefits from regularity
- AMR brings this down to about $O(100)$ TB
  - furthermore, full GR only necessary for the very, very final stages
- and the data can be processed on the fly (*big data without big data*)
- other GR applications similar

LIGO, LHC, Euclid, SKA, . . .

Streaming Data Analytics

- structured and mostly "small" independent chunks
  - distributing by chunk natural and trivial
- traditional HPC approaches still work
  - but may need an unprecedented scale

Data Archiving

- largely a solved problem
  - again unprecedented scale but scales easily
- good metadata needed to access data later
  - long history of metadata in astronomy, spanning over 100 years

Easy

- this is all known data: we know what we are looking for
- plans to allow later reuse to look for the unknown

Google's PageRank

Getting Data In to PageRank

- this is not just "unstructured" data in the above sense, but it actually has no structure at all before it's processed
- stringy data not nice for CPUs and not structured at all
- a collection of web pages have
  - different numbers of links per page
  - different page sizes
  - different links are of different length
- lots of I/O (even ignoring the web crawl)
  - but time to crawl a website » time to process it
- etc
- fortunately almost embarrassingly parallel
- one-time cost per page:
  - data from pages already processed unaffected by new data

PageRank

What does it do?

- eventually this solves R from

$$R = dMR + \frac{1-d}{N}$$

  where N is the number of pages on the Web and the matrix M is NxN, and $d \in (0,1)$ is a "damping" factor
- $N > 1000000000$ so no way to solve directly
- M is sparse and iterative algorithms converge quickly because M is a stochastic matrix
- sparse linear algebra well understood, widely used, less complex than dense
  - but is unfriendly to the CPU and memory, hurting efficiency

More examples of unstructured problems

Databases

- Seems stuck in non-distributed era (for a reason?)
- Searching for needles in haystacks is much worse than indirect access
  - there are tricks, like hash tables, directory caches etc

Graph searches

- typically involves pointer chasing: indirect access again
- software prefetching on KNL helps when access pattern is known to the programmer (but undecipherable to the prefetcher)
- won't say much more since there's a talk on this later!

Neural Nets

- not sure how HPC folks can help here: suggestions?
- but perhaps best discussed later: there's a talk on neural nets later