A UWS service to cross-match (very) large catalogues

Thomas Boch, **François-Xavier Pineau**, Sébastien Derrière
and Brice Gassmann

CDS, Observatoire Astronomique de Strasbourg

IVOA Interop, Nara, 07 December 2010

Context

CDS cross-match service (in development)

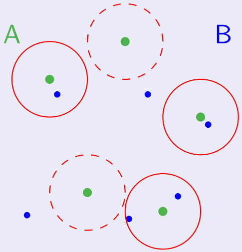
- Based on UWS (job submission)
- Catalogues :
- Algorithms :

- Particularity : deal with (very) large catalogues

Context

CDS cross-match service (in development)

- Based on UWS (job submission)
- Catalogues :
- Algorithms :

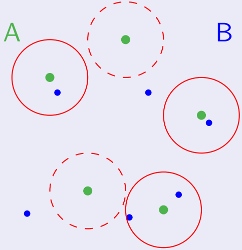


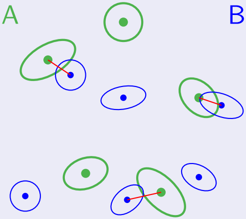
- Particularity : deal with (very) large catalogues

Context

CDS cross-match service (in development)

- Based on UWS (job submission)
- Catalogues :
- Algorithms :



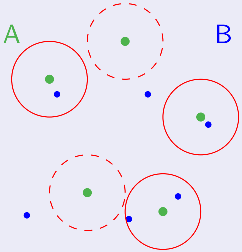


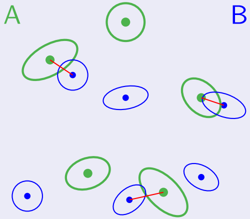
- Particularity : deal with (very) large catalogues

Context

CDS cross-match service (in development)

- Based on UWS (job submission)
- Catalogues :
- Algorithms :





- Particularity : deal with (very) large catalogues

Dealing with (very) large catalogues

Example

- 2MASS
  - $\sim 470\text{x}10^6$ sources
  - minimal data $\sim$ 15 GB
    - identifier (integer 4 Bytes)
    - positions (double 8 B+8 B)
    - errors (float 4 B+4 B+4 B)
- USNO-B1
  - $\sim 10^9$ sources
  - minimal data $\sim$ 28 GB
    - identifier (integer 4 B)
    - positions (double 8 B+8 B)
    - errors (float 4 B+4 B)
- LSST projection at 5 years:
  - V>26, $\sim 3\text{x}10^9$ **unique** sources
  - minimal data $\sim$ 96 GB

Problems

- Data size
  - do not fit into memory
- Performance issues
  - data loading
  - looking for candidates

Solutions

- Scalability: Healpix partitioning
- Efficiency:
  - special indexed binary file
  - *kd*-tree (cone search queries)
  - multithreading
  - parallel processing

Dealing with (very) large catalogues

Example

- 2MASS
  - $\sim$ 470x10^6 sources
  - minimal data $\sim$ 15 GB
    - identifier (integer 4 Bytes)
    - positions (double 8 B+8 B)
    - errors (float 4 B+4 B+4 B)
- USNO-B1
  - $\sim 10^9$ sources
  - minimal data $\sim$ 28 GB
    - identifier (integer 4 B)
    - positions (double 8 B+8 B)
    - errors (float 4 B+4 B)
- LSST projection at 5 years:
  - V>26, $\sim$ 3x10^9 **unique** sources
  - minimal data $\sim$ 96 GB

Problems

- Data size
  - do not fit into memory
- Performance issues
  - data loading
  - looking for candidates

Solutions

- Scalability: Healpix partitioning
- Efficiency:
  - special indexed binary file
  - *kd*-tree (cone search queries)
  - multithreading
  - parallel processing

Dealing with (very) large catalogues

Example

- 2MASS
  - $\sim 470\text{x}10^6$ sources
  - minimal data $\sim$ 15 GB
    - identifier (integer 4 Bytes)
    - positions (double 8 B+8 B)
    - errors (float 4 B+4 B+4 B)
- USNO-B1
  - $\sim 10^9$ sources
  - minimal data $\sim$ 28 GB
    - identifier (integer 4 B)
    - positions (double 8 B+8 B)
    - errors (float 4 B+4 B)
- LSST projection at 5 years:
  - V>26, $\sim 3\text{x}10^9$ **unique** sources
  - minimal data $\sim$ 96 GB

Problems

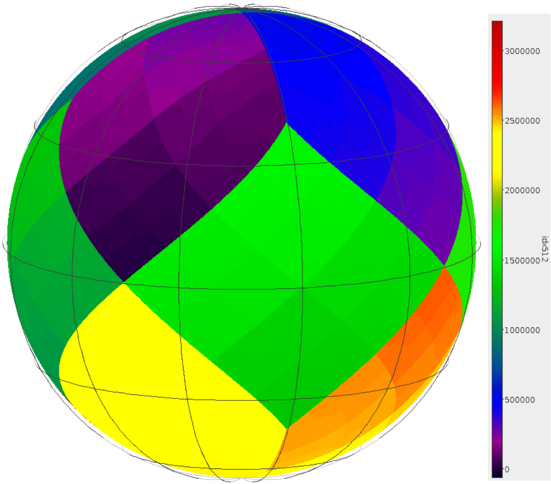
- Data size
  - do not fit into memory
- Performance issues
  - data loading
  - looking for candidates

Solutions

- Scalability: Healpix partitioning
- Efficiency:
  - special indexed binary file
  - *kd*-tree (cone search queries)
  - multithreading
  - parallel processing

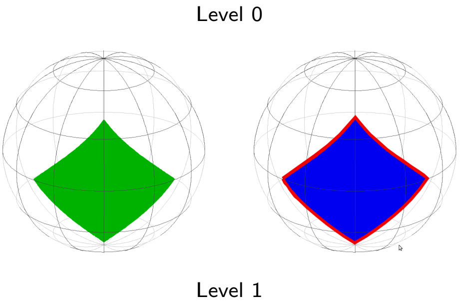
Healpix

- Hierarchical sky pixelisation
  - level 0 $\rightsquigarrow$ 12 pixels
  - level 1 $\rightsquigarrow$ 12x4 pixels
  - ...
  - level n $\rightsquigarrow$ $12x2^{2n}$
- Pixels of equal area
- Developed at NASA: `healpix.jpl.nasa.gov`
- Available in
  - C, C++
  - Fortran
  - IDL
  - Java
  - ...?



Scalable cross-match

- Independent pixels cross-match
  - but border effects
- Cat. B pixel sources put in a kd-tree
- Optimal partitioning level
  - available memory
  - minimisation of:
    $$\sum_{i=0}^{nPixels} N_{A_i} \log(1 + N_{B_i} + N_{B_i}^{b})$$
  - I/O cost

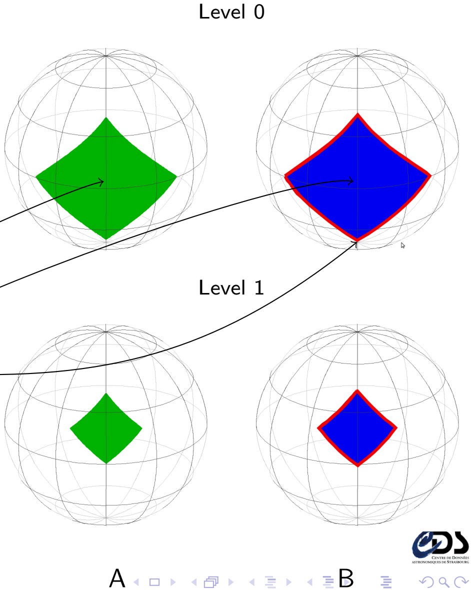


Scalable cross-match

- Independent pixels cross-match
  - but border effects
- Cat. B pixel sources put in a kd-tree
- Optimal partitioning level
  - available memory
  - minimisation of:

$$\sum_{i=0}^{nPixels} \boxed{N_{A_i}} \log(1 + \boxed{N_{B_i}} + \boxed{N_{B_i}^b})$$

  - I/O cost



Scalable cross-match

Single machine

- All sky correlation (small catalogues)
  - allow “on the fly” correlation
- Correlation pixel by pixel (large catalogues)

Computer grid

- Parallel processing
  - Distribute job pieces on separate machines
- “On the fly” correlation possible

Scalable cross-match

Single machine

- All sky correlation (small catalogues)
  - allow “on the fly” correlation
- Correlation pixel by pixel (large catalogues)

Computer grid

- Parallel processing
  - Distribute job pieces on separate machines
- “On the fly” correlation possible

Scalable cross-match

Single machine

- All sky correlation (small catalogues)
  - allow "on the fly" correlation
- Correlation pixel by pixel (large catalogues)

Computer grid

- Parallel processing
  - Distribute job pieces on separate machines
- "On the fly" correlation possible

Scalable cross-match

Single machine

- All sky correlation (small catalogues)
  - allow "on the fly" correlation
- Correlation pixel by pixel (large catalogues)

Computer grid

- Parallel processing
  - Distribute job pieces on separate machines
- "On the fly" correlation possible

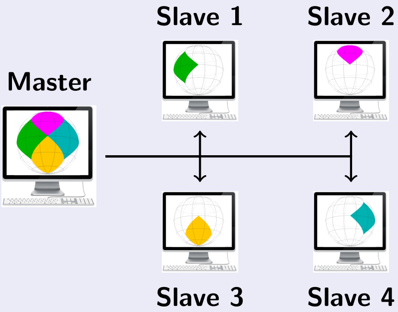
Scalable cross-match

Single machine

- All sky correlation (small catalogues)
  - allow “on the fly” correlation
- Correlation pixel by pixel (large catalogues)

Computer grid

- Parallel processing
  - Distribute job pieces on separate machines
- “On the fly” correlation possible



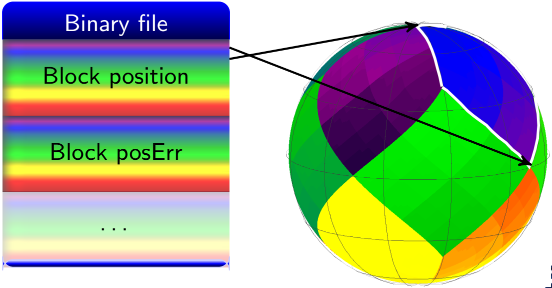
Loading data: indexed binary files

Index files

- One by healpix level
- For each pixel
  - offset
  - nSources

Binary data file

- Organized by blocks:
  - positions
  - position errors
  - identifiers
  - ...
- Sources ordered by healpix pixel index



Loading data: indexed binary files

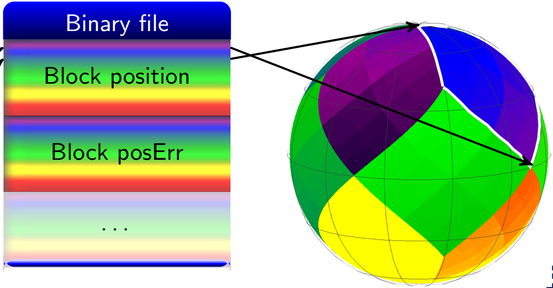
Index files

- One by healpix level
- For each pixel
  - offset
  - nSources

Binary data file

- Organized by blocks:
  - positions
  - position errors
  - identifiers
  - ...
- Sources ordered by healpix pixel index

level 0 index file		
Idx	offset	nSrcs
0	xxx	xxx
1	xxx	xxx
⋮	⋮	⋮
11	xxx	xxx



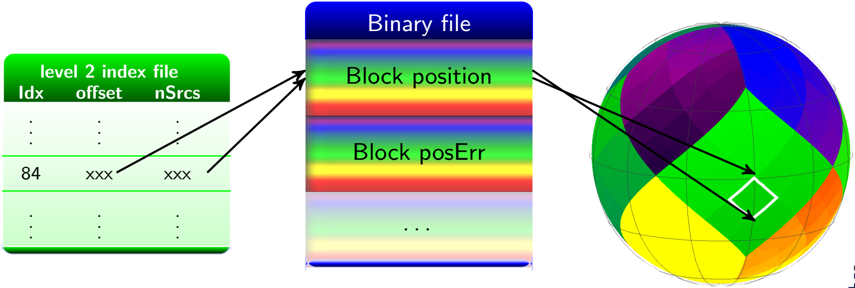
Loading data: indexed binary files

Index files

- One by healpix level
- For each pixel
  - offset
  - nSources

Binary data file

- Organized by blocks:
  - positions
  - position errors
  - identifiers
  - ...
- Sources ordered by healpix pixel index



*k*d-tree

What is a *k*d-Tree?

- A space-partitioning data structure
- Allows for fast k-nearest neighbour/cone search queries
  - nearest neighbour query in $O(\log(n))$

Problem

- Naive implementation can be memory consuming
- We want a memory efficient *k*d-tree (capacity $>$ 1 billion sources)

Solution

- To use a single array (sorted using a *k*d-tree scheme)

kd-tree

What is a kd-Tree?

- A space-partitioning data structure
- Allows for fast k-nearest neighbour/cone search queries
  - nearest neighbour query in $O(\log(n))$

Problem

- Naive implementation can be memory consuming
- We want a memory efficient kd-tree (capacity $>$ 1 billion sources)

Solution

- To use a single array (sorted using a kd-tree scheme)

kd-tree

What is a kd-Tree?

- A space-partitioning data structure
- Allows for fast k-nearest neighbour/cone search queries
  - nearest neighbour query in $O(\log(n))$

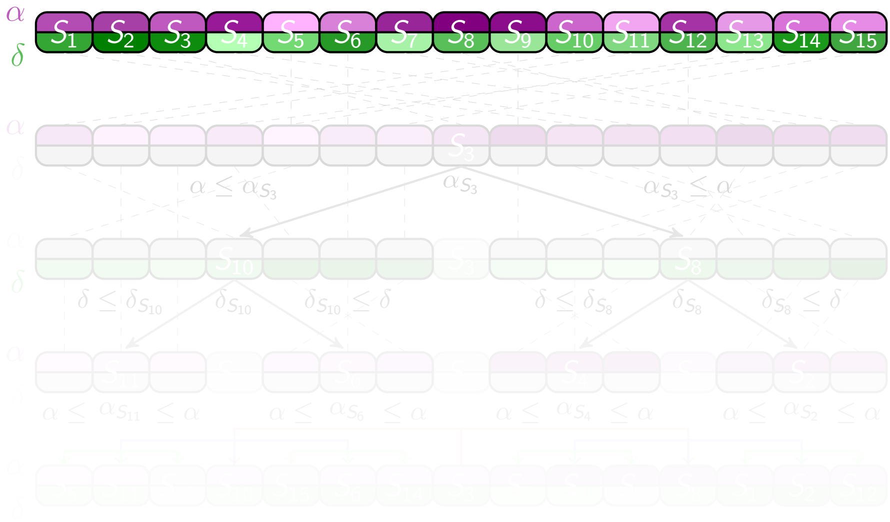
Problem

- Naive implementation can be memory consuming
- We want a memory efficient kd-tree (capacity $>$ 1 billion sources)

Solution

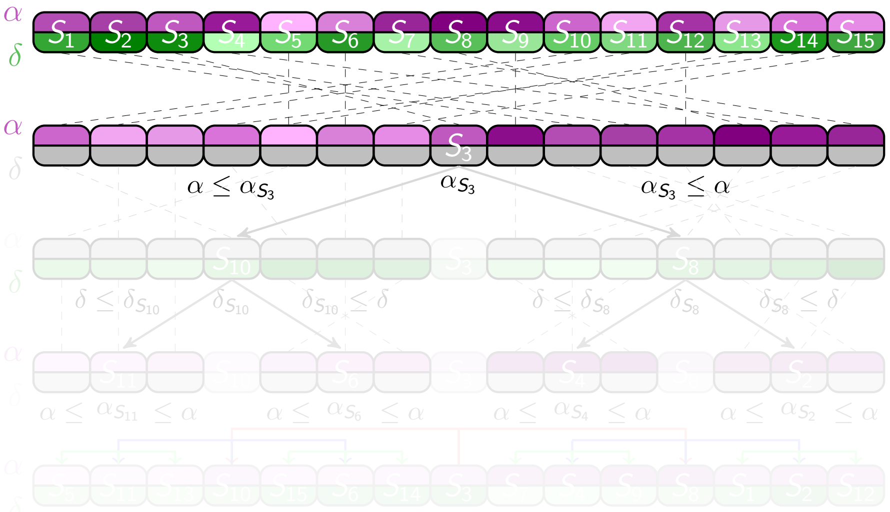
- To use a single array (sorted using a kd-tree scheme)

A *kd*-tree can be a simple sorted array of sources
Algorithm: *quicksort* alternating the sorted coordinate

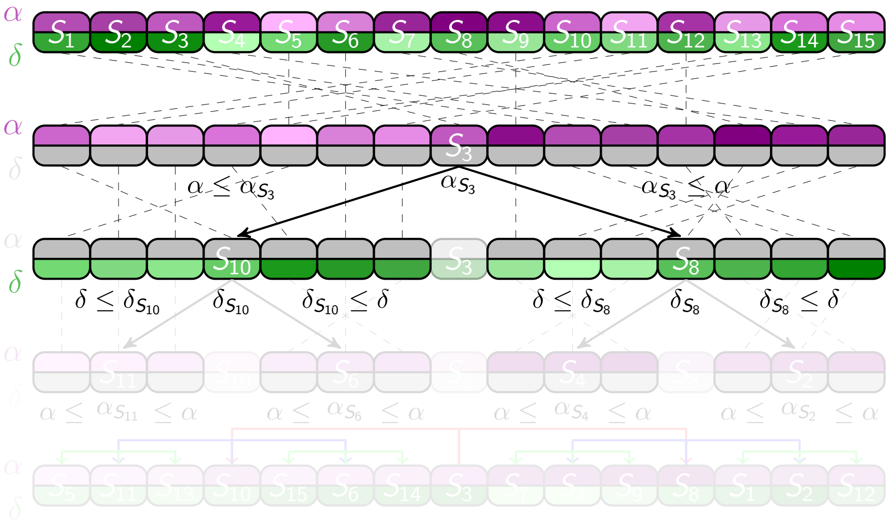


A *kd*-tree can be a simple sorted array of sources

Algorithm: *quicksort* alternating the sorted coordinate

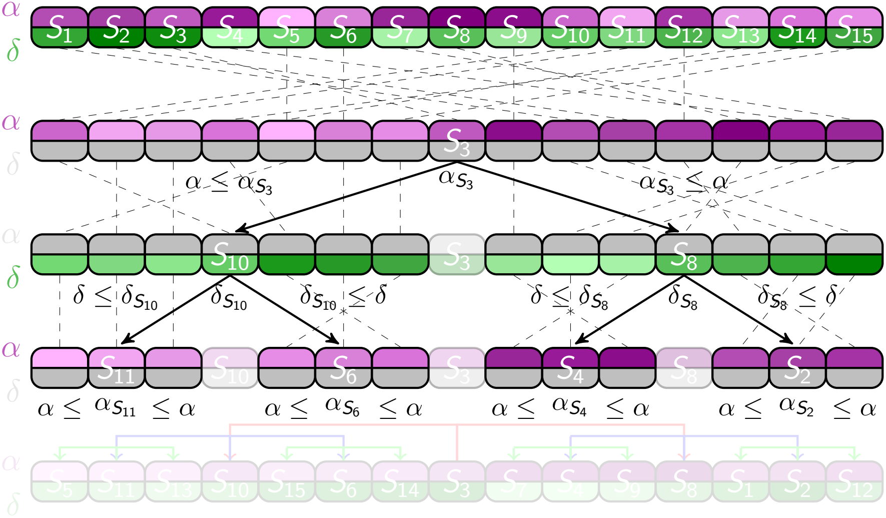


A *kd*-tree can be a simple sorted array of sources
Algorithm: *quicksort* alternating the sorted coordinate



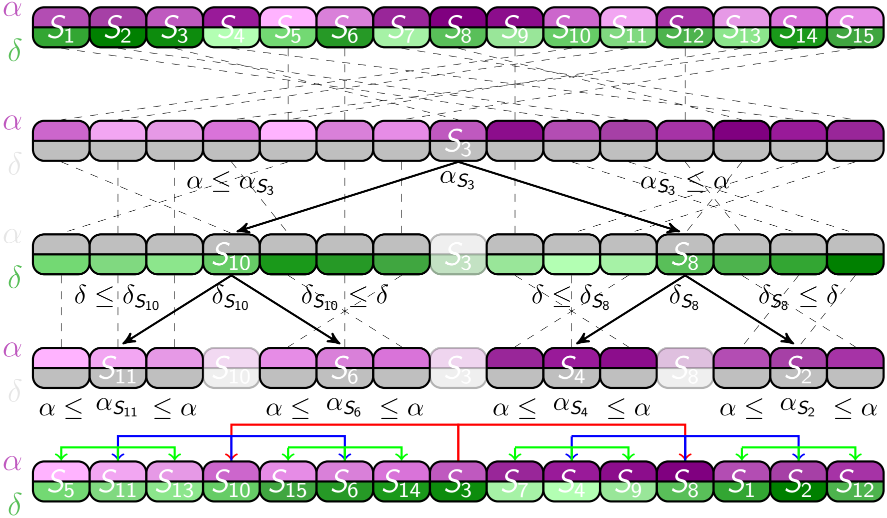
A *kd*-tree can be a simple sorted array of sources

Algorithm: *quicksort* alternating the sorted coordinate



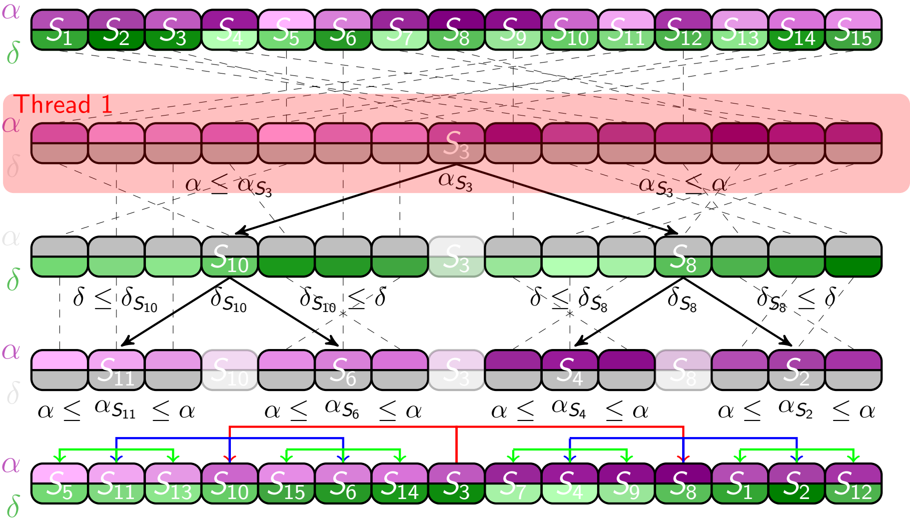
Creation speed up by using multi-threading

A kd-tree can be a simple sorted array of sources
Algorithm: *quicksort* alternating the sorted coordinate



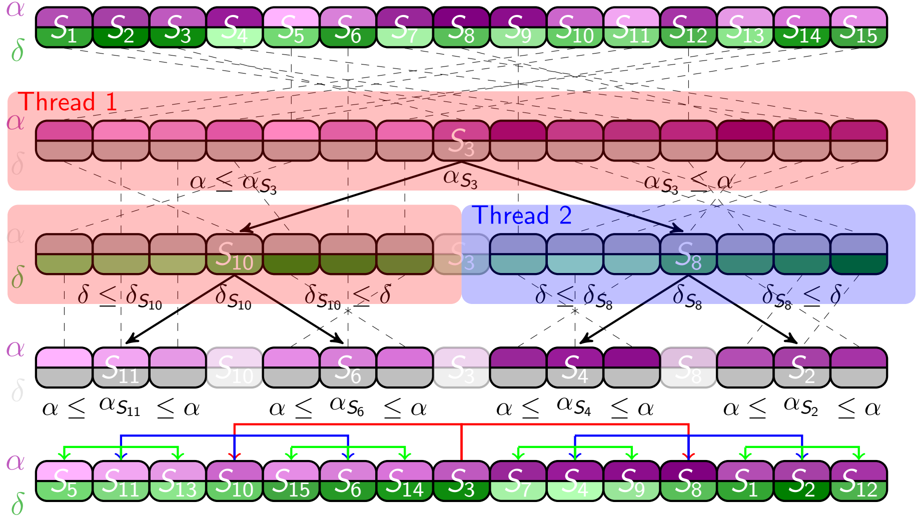
Creation speed up by using multi-threading

A kd-tree can be a simple sorted array of sources
Algorithm: *quicksort* alternating the sorted coordinate



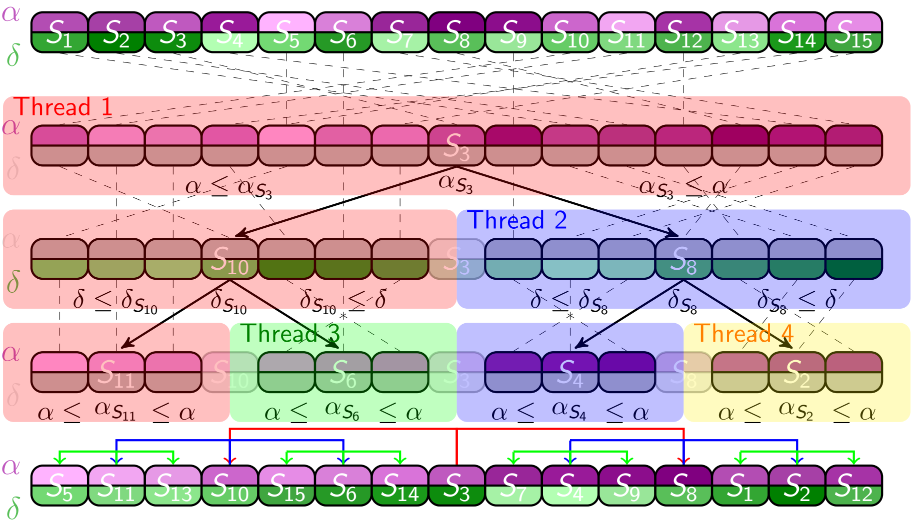
Creation speed up by using multi-threading

A kd-tree can be a simple sorted array of sources
Algorithm: *quicksort* alternating the sorted coordinate



Creation speed up by using multi-threading

A kd-tree can be a simple sorted array of sources
Algorithm: *quicksort* alternating the sorted coordinate



Creation speed up by using multi-threading

Modified *k*d-tree and multithreading

Modified *k*d-tree

- Classical *k*d-tree adapted for euclidian spaces
- Solution 1: (rejected)
  - cartesian coordinates (x, y, z)
    - $\rightsquigarrow$ time consuming (conversion)
    - $\rightsquigarrow$ memory consuming (+50%)
- Solution 2: (approved)
  - spherical coordinates (α, δ)
  - classical creation algorithm
  - modified query algorithm
    - angular distances (Haversine formula)
    - modified circle/rectangle intersection to enter a sub-tree

Multithreading

- Single *k*NN or cone search query not multithread
- Pool of `threads` executing multiple queries simultaneously

Modified *k*d-tree and multithreading

Modified *k*d-tree

- Classical *k*d-tree adapted for euclidian spaces
- Solution 1: (rejected)
  - cartesian coordinates (x, y, z)
    - $\rightsquigarrow$ time consuming (conversion)
    - $\rightsquigarrow$ memory consuming (+50%)
- Solution 2: (approved)
  - spherical coordinates (α, δ)
  - classical creation algorithm
  - modified query algorithm
    - angular distances (Haversine formula)
    - modified circle/rectangle intersection to enter a sub-tree

Multithreading

- Single *k*NN or cone search query not multithread
- Pool of `threads` executing multiple queries simultaneously

Modified *k*d-tree and multithreading

Modified *k*d-tree

- Classical *k*d-tree adapted for euclidian spaces
- Solution 1: (rejected)
  - cartesian coordinates (x, y, z)
    - $\rightsquigarrow$ time consuming (conversion)
    - $\rightsquigarrow$ memory consuming (+50%)
- Solution 2: (approved)
  - spherical coordinates (α, δ)
  - classical creation algorithm
  - modified query algorithm
    - angular distances (Haversine formula)
    - modified circle/rectangle intersection to enter a sub-tree

Multithreading

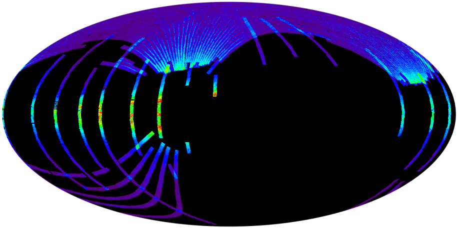
- Single *k*NN or cone search query not multithread
- Pool of `threads` executing multiple queries simultaneously

Test Machine

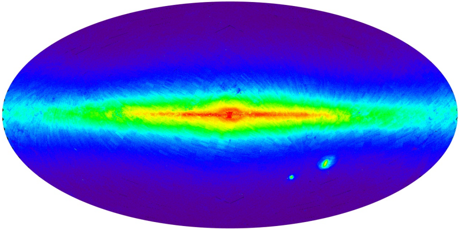
- Dell machine 2 600€(∼$3 600):
  - 24 GB of **1333 MHz** memory
  - 2x Quad Core 2.27 GHz (Xeon)
  - **16 threads** (Hyper-Threading)
  - High speed HDD (10 000 rpm)

Test results

Full catalogue cross-correlation



SDSS DR7 (∼357 000 000 sources)

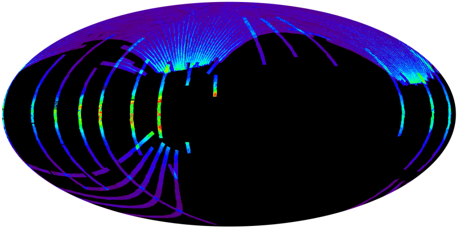


2MASS (∼470 000 000 sources)

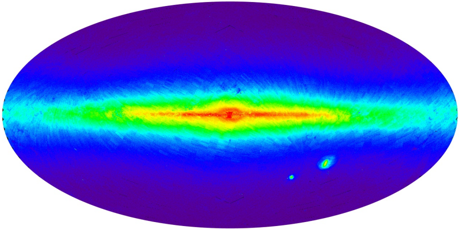
- Simple cross-match: **∼9 min**
  - radius of 5″
  - Healpix level 3 (∼7.3°)
  - Level 9 borders (∼7')
  - ∼49 209 000 associations
- With elliptical errors: **∼10 min**
  - distance of 3.44σ
  - distance max of 5″
  - Healpix level 3
  - ∼37 507 000 associations

Test results

Full catalogue cross-correlation



SDSS DR7 (∼357 000 000 sources)



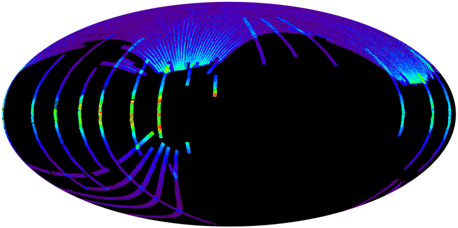
2MASS (∼470 000 000 sources)

- Simple cross-match: **∼9 min**
  - radius of 5″
  - Healpix level 3 (∼7.3°)
  - Level 9 borders (∼7')
  - ∼49 209 000 associations

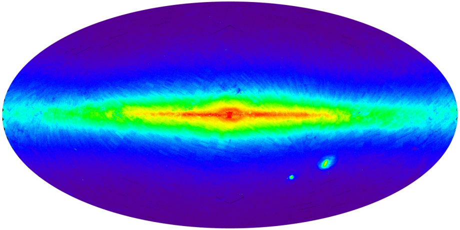
- With elliptical errors: **∼10 min**
  - distance of 3.44σ
  - distance max of 5″
  - Healpix level 3
  - ∼37 507 000 associations

Test results

Full catalogue cross-correlation



SDSS DR7 (∼357 000 000 sources)

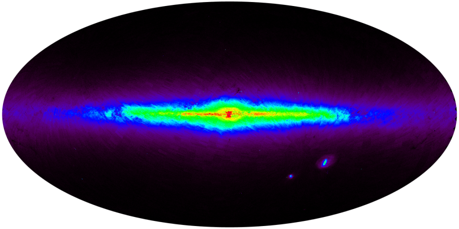


2MASS (∼470 000 000 sources)

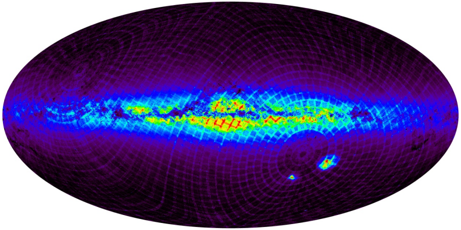
- Simple cross-match: **∼9 min**
  - radius of $5''$
  - Healpix level 3 (∼7.3°)
  - Level 9 borders (∼7')
  - ∼49 209 000 associations
- With elliptical errors: **∼10 min**
  - distance of 3.44σ
  - distance max of $5''$
  - Healpix level 3
  - ∼37 507 000 associations

Test results

Full all-sky catalogues cross-correlation



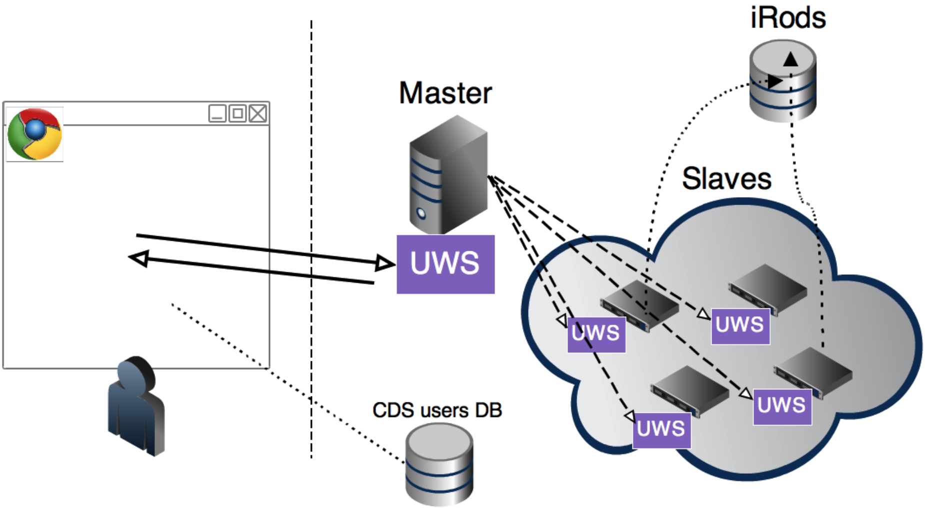
2MASS (∼470 000 000 sources)



USNO-B1 (∼1 046 000 000 sources)

- Simple cross-match: **∼30 min**
  - radius of 5′′
  - Healpix level 3
  - Level 9 borders
  - ∼583 300 000 associations

General architecture

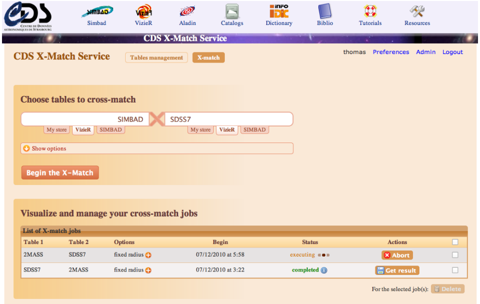


UWS layer

- Provided by a Java library developed at CDS by Grégory Mantelet
  - Documentation and tutorial : `http://saada.u-strasbg.fr/uwstuto/index.html`
  - Distributed under LPGL licence
  - Will be presented at GWS2 session on Friday
- Internal UWS enables communication between master and slaves

Web interface

- Simple front-end to access the UWS interface
- Job submission, retrieval of jobs status through **JSON calls**
- Integrated with the CDS login service (used for the Annotations and the Portal)
  - allow users to upload (and cross-match) their own tables
- Demonstration



Lessons learned

Hardware

For our application:

- RAM frequency **does** matter (lots of memory access)
- Hyper-Threading **does** matter (on 8 cores, 16 threads ∼2x faster than 8 threads)

Software: don't have *a priori*

- Efficient full Java code
- Efficient modifed *k*d-trees (in our case)

Service

- Existing and future (very) large catalogues can be processed
- Bottleneck is data transfer (without surprise)
  - service colocated with data