Efficient and scalable cross-matching of (very) large catalogues

François-Xavier Pineau[1], Thomas Boch[1] and Sebastien Derrière[1]

[1]CDS, Observatoire Astronomique de Strasbourg

ADASS Boston, 08 November 2010

Context

CDS cross-match service (in development)

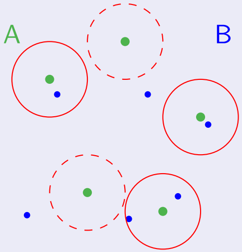
- Based on UWS (job submission)
- Catalogues :
- Algorithms :

- Particularity : deal with (very) large catalogues

Context

CDS cross-match service (in development)

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

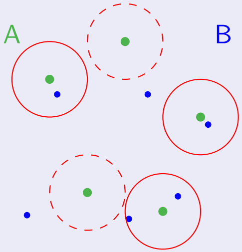


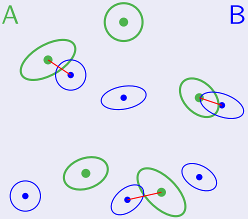
- Particularity : deal with (very) large catalogues

Context

CDS cross-match service (in development)

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



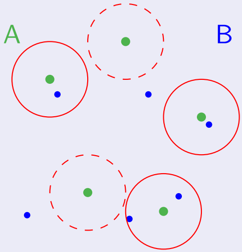


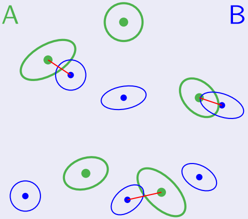
- Particularity : deal with (very) large catalogues

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
  - $\sim$ 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
  - $\sim$ 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
  - $\sim$ 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

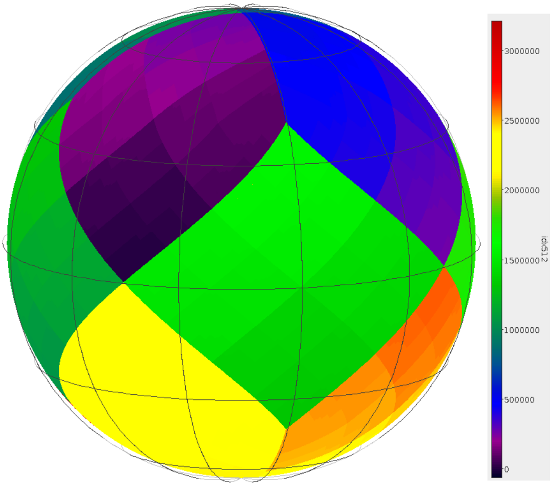
- Data size
  - do not fit into memory
- Performance issues
  - data loading
  - looking for candidates

Solutions

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

Healpix

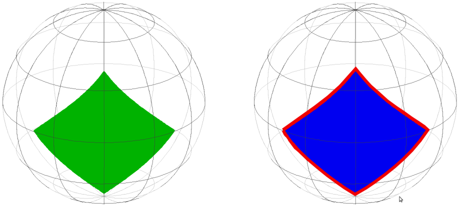
- Hierarchical sky pixelisation
  - level 0 $\rightsquigarrow$ 12 pixels
  - level 1 $\rightsquigarrow$ 12x4 pixels
  - ...
  - level n $\rightsquigarrow$ $12 \mathrm{x} 2^{2n}$
- Pixels of equal area
- Developed at NASA: `healpix.jpl.nasa.gov`
- Available in
  - C, C++
  - Fortran
  - IDL
  - Java
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



Scalable cross-match

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

Level 0



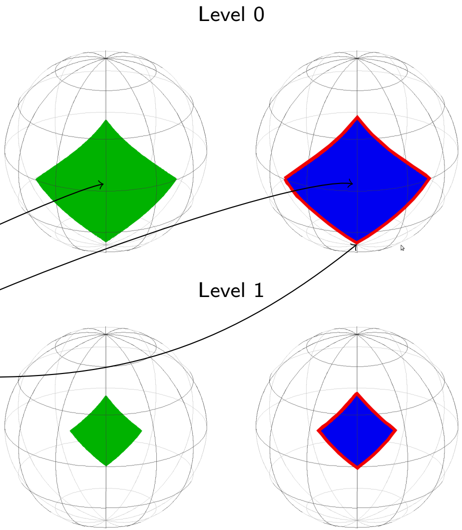
Level 1

Scalable cross-match

- Independent pixels cross-match
  - but border effects
- Cat. B pixel sources put in a *k*d-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

Single machine

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

Computer grid

- Parallel processing
- Framework:
  - based on UWS[a] (few machines)
  - Hadoop (large grid)
- “On the fly” correlation possible

[a]Universal Worker Service (IVOA)

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
- Framework:
  - based on UWS[a] (few machines)
  - Hadoop (large grid)
- "On the fly" correlation possible

[a] Universal Worker Service (IVOA)

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
- Framework:
  - based on UWS[a] (few machines)
  - Hadoop (large grid)
- "On the fly" correlation possible

[a] Universal Worker Service (IVOA)

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
- Framework:
  - based on UWS[a] (few machines)
  - Hadoop (large grid)
- "On the fly" correlation possible

[a] Universal Worker Service (IVOA)

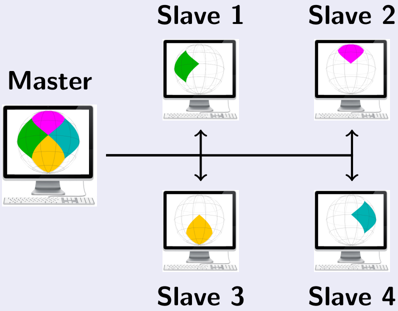
Scalable cross-match

Single machine

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

Computer grid

- Parallel processing
- Framework:
  - based on UWS[a] (few machines)
  - Hadoop (large grid)
- "On the fly" correlation possible



[a]Universal Worker Service (IVOA)

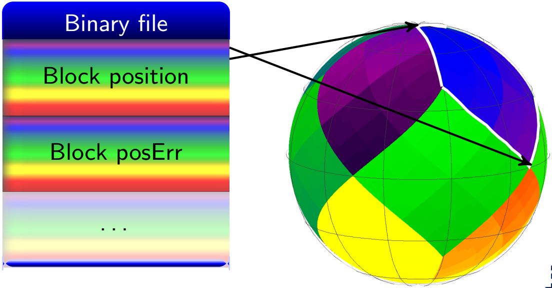
Loading data: indexed binary files

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

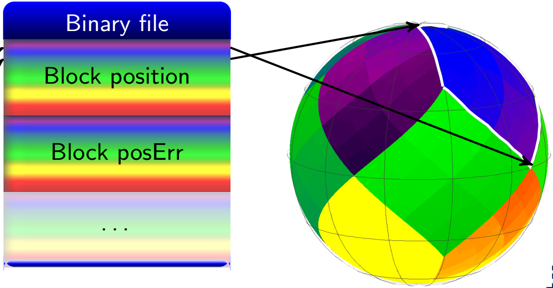
- One by healpix level
- For each pixel
  - offset
  - nSources

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



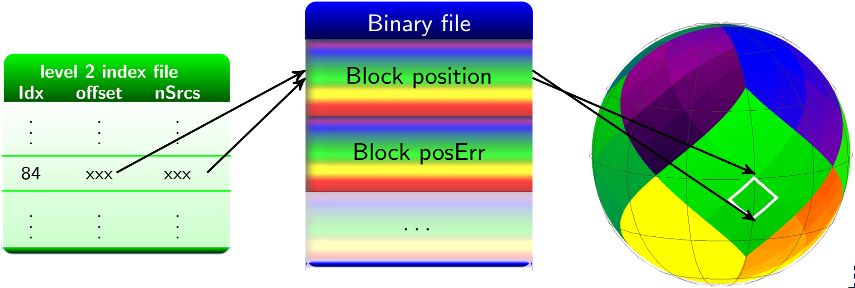
Loading data: indexed binary files

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 $O(\log(n))$

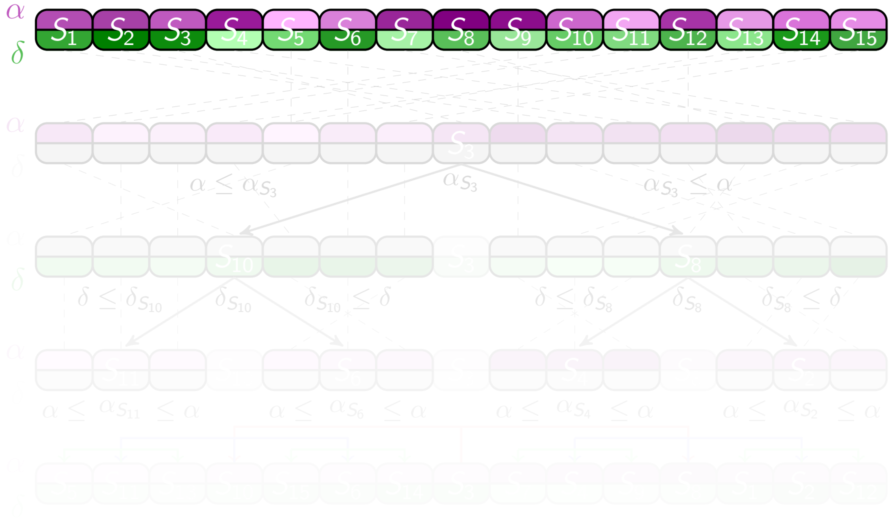
Problem

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

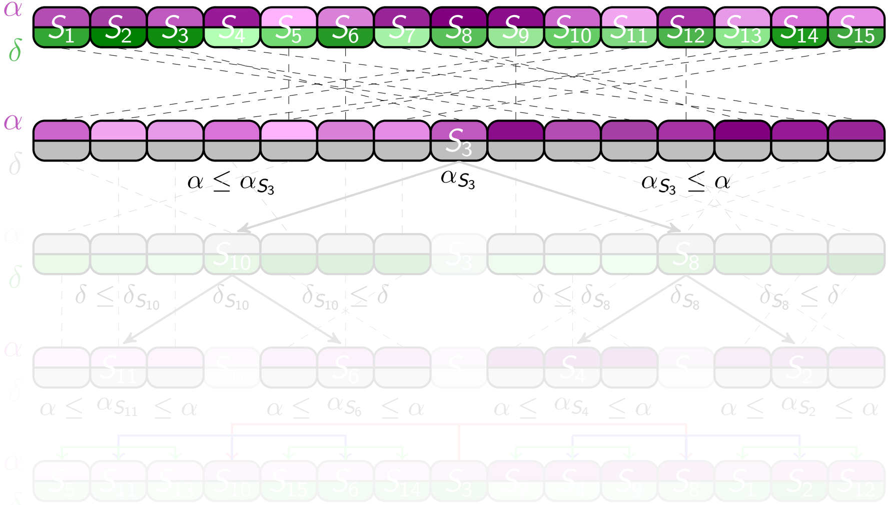
Solution

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

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

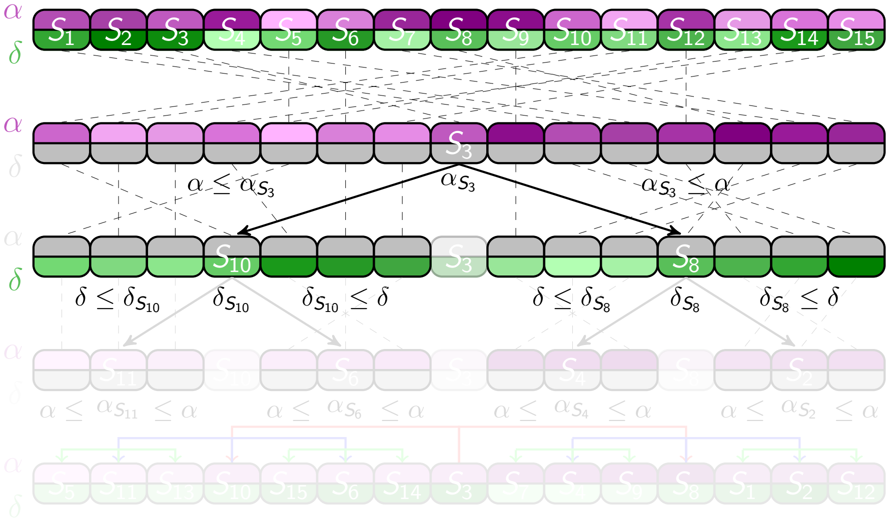


A *kd*-tree can be a simple sorted array of sources
Algorithm: *quicksort* alternating the sorted coordinate



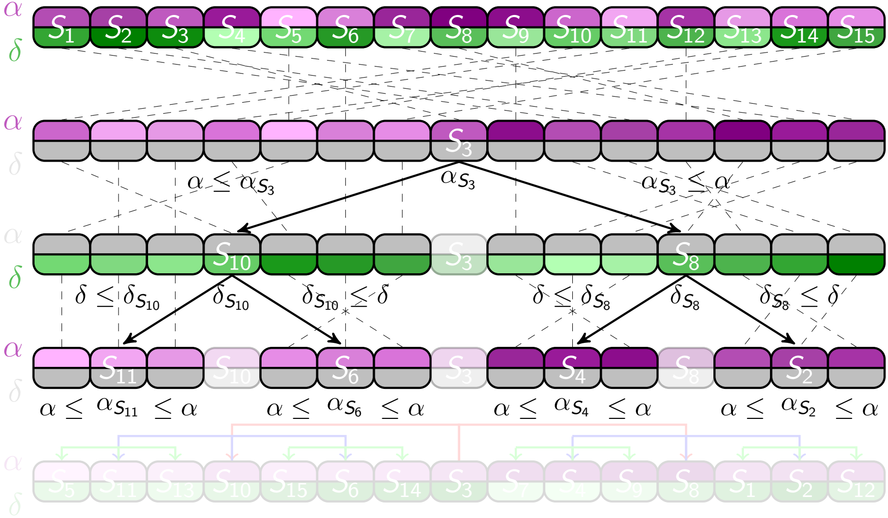
A *kd*-tree can be a simple sorted array of sources

Algorithm: *quicksort* alternating the sorted coordinate



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

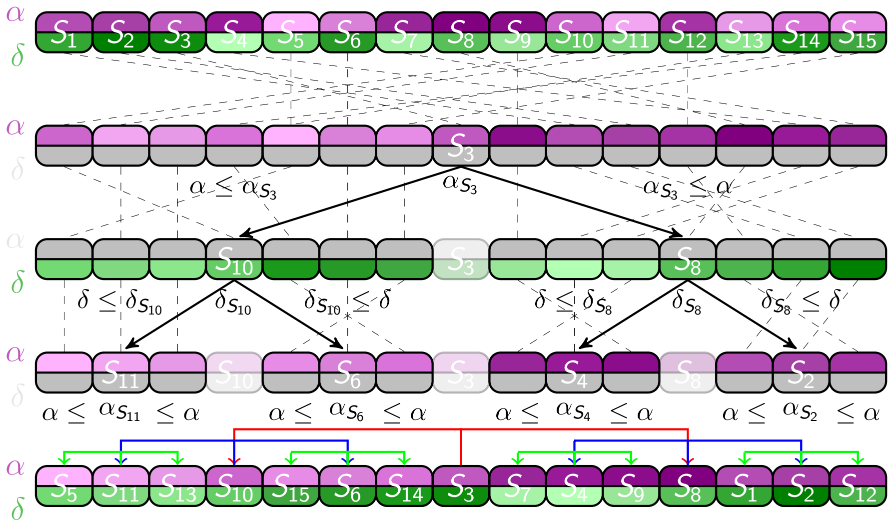
Algorithm: *quicksort* alternating the sorted coordinate



Creation speed up by using multi-threading

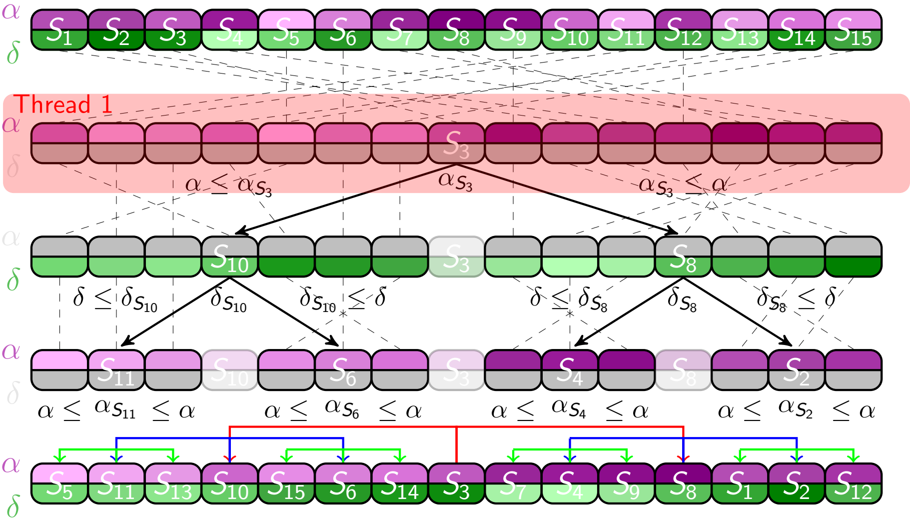
A *kd*-tree can be a simple sorted array of sources

Algorithm: *quicksort* alternating the sorted coordinate



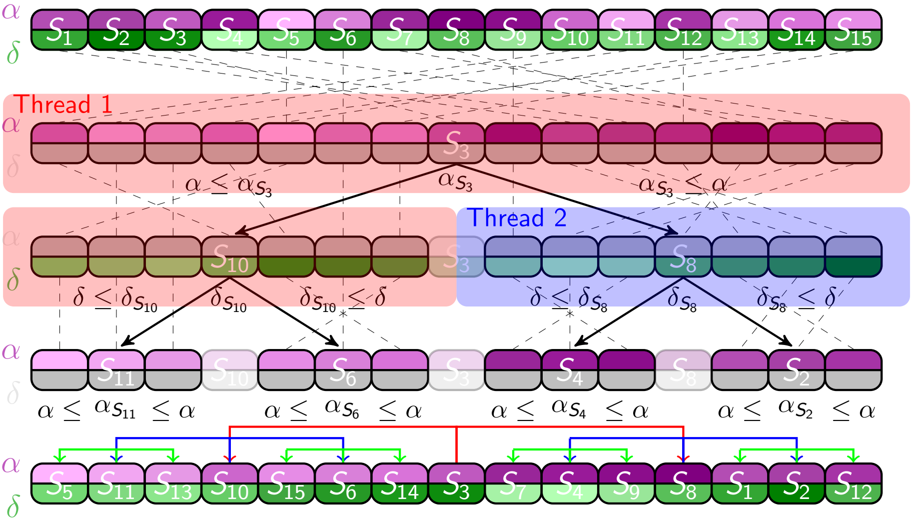
Creation speed up by using multi-threading

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



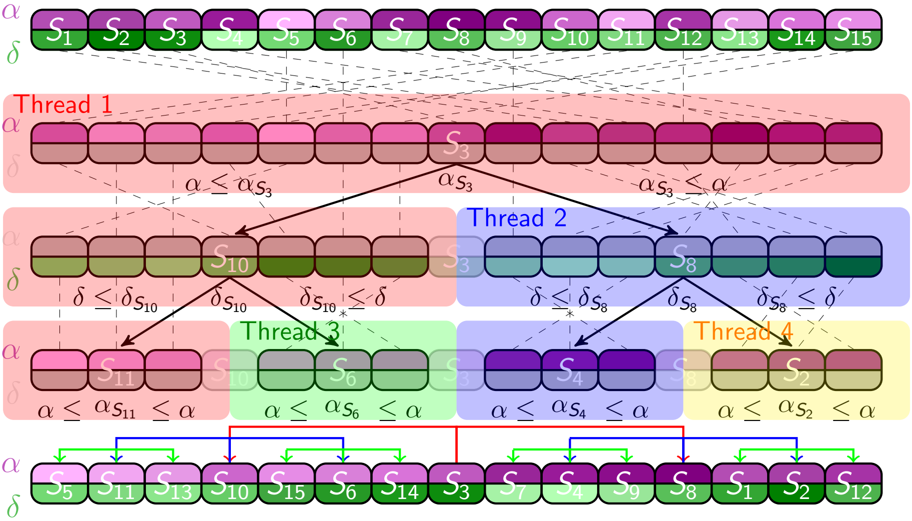
Creation speed up by using multi-threading

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

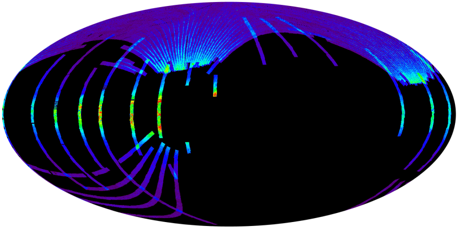
- Single *k*NN or cone search query not multithread
- Pool of `threads` executing multiple queries simultaneously

Test Machine

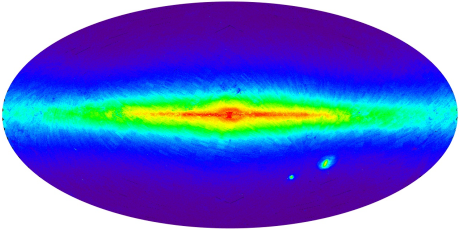
- Dell machine 2 600€(∼$3 600):
  - 24 GB of **1333 MHz** memory
  - 2x Quad Core 2.27 GHz (Xeon)
  - **16 threads** (Hyper-Threading)
  - High speed HDD (10 000 rpm)

Test results

Full catalogue cross-correlation



SDSS DR7 (∼357 000 000 sources)

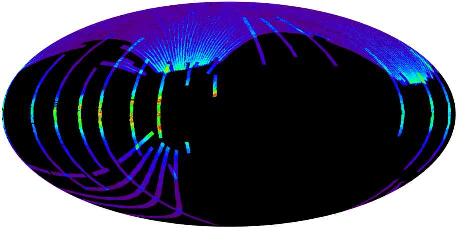


2MASS (∼470 000 000 sources)

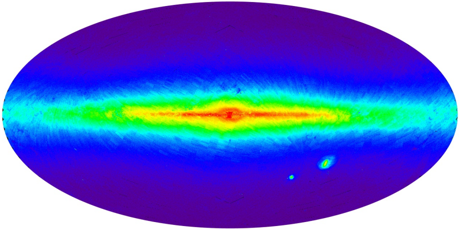
- Simple cross-match: **∼9 min**
  - radius of 5″
  - Healpix level 3 (∼7.3°)
  - Level 9 borders (∼7')
  - ∼49 209 000 associations
- With elliptical errors: **∼10 min**
  - distance of 3.44σ
  - distance max of 5″
  - Healpix level 3
  - ∼37 507 000 associations

Test results

Full catalogue cross-correlation



SDSS DR7 ($\sim$357 000 000 sources)

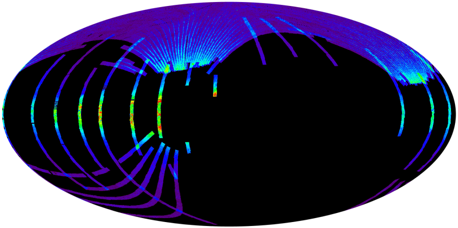


2MASS ($\sim$470 000 000 sources)

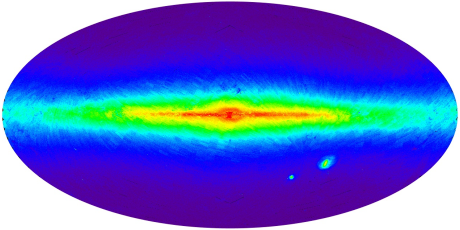
- Simple cross-match: **$\sim$9 min**
  - radius of $5''$
  - Healpix level 3 ($\sim$7.3°)
  - Level 9 borders ($\sim$7')
  - $\sim$49 209 000 associations
- With elliptical errors: **$\sim$10 min**
  - distance of 3.44σ
  - distance max of $5''$
  - Healpix level 3
  - $\sim$37 507 000 associations

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

Lessons learned

Hardware

For our application:

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

Software: don't have *a priori*

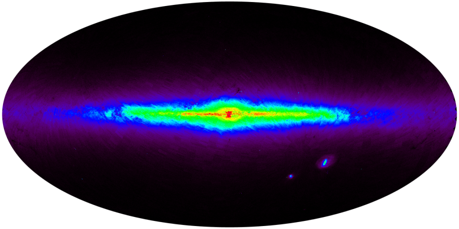
- Efficient full Java code
- Efficient modifed *k*d-trees (in our case)

Service

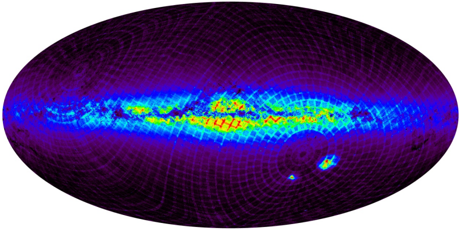
- Existing and future (very) large catalogues can be processed
- Bottleneck is data transfer (without surprise)
  - service colocated with data

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

Basic likelihood ratio (LR)

Ratio between:

- Rayleigh distribution

$$LR = \frac{\boxed{r \exp\left(-\frac{1}{2}r^2\right)}}{2\lambda r} = \frac{\exp\left(-\frac{1}{2} r^2\right)}{2\lambda}$$

- Poissonian distribution

Depends on:

- r = normalized distance in σ
- $\lambda \propto$ local density of sources

Test results

- SDSS7 x 2MASS correlations + LRs
- Local densities estimated by kNN averaging (k=100)
- $\rightsquigarrow$ 15min

Basic likelihood ratio (LR)

Ratio between:

- Rayleigh distribution
- Poissonian distribution

$$LR = \frac{\boxed{r \exp\left(-\frac{1}{2}r^2\right)}}{2\lambda r} = \frac{\exp\left(-\frac{1}{2}\boxed{r}^2\right)}{2\boxed{\lambda}}$$

Depends on:

- r = normalized distance in σ
- $\lambda \propto$ local density of sources

Test results

- SDSS7 x 2MASS correlations + LRs
- Local densities estimated by kNN averaging (k=100)
- $\rightsquigarrow$ 15min

Basic likelihood ratio (LR)

Ratio between:

- Rayleigh distribution
- Poissonian distribution

$$LR = \frac{\boxed{r \exp\left(-\frac{1}{2} r^2\right)}}{2\lambda r} = \frac{\exp\left(-\frac{1}{2}\, r^2\right)}{2\,\lambda}$$

Depends on:

- $r =$ normalized distance in σ
- $\lambda \propto$ local density of sources

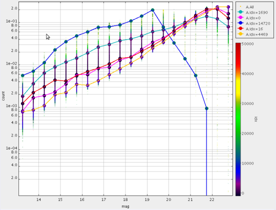
Test results

- SDSS7 x 2MASS correlations + LRs
- Local densities estimated by kNN averaging (k=100)
- $\rightsquigarrow$ 15min

Going further...

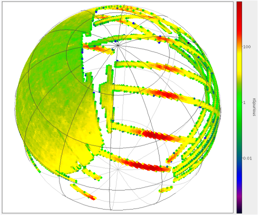
Magnitude-dependent LRs (fast solution)

- $\rho(m \pm \Delta m) = \rho \; \mathrm{p}(m \pm \Delta m)$
- kNN averaging
- log N-log S law
- SDSS7, level 6 ($\sim 1^\circ$), 15 187 non empty histograms computed in 30s.



Probability of identifications (fast solution)

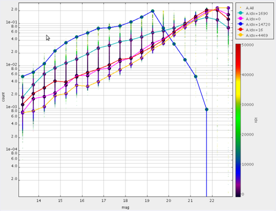
- Number of spurious match estimates
  - Positional errors sampling for both catalogues
  - $N_{spur} = \sum_A \sum_B S_{conv} / S_{pixel}$
- SDSS7 x 2MASS, level 6, 8min (not yet multithreaded!)



Going further...

Magnitude-dependent LRs (fast solution)

- $\rho(m \pm \Delta m) = \rho\ \mathrm{p}(m \pm \Delta m)$
- kNN averaging
- log N-log S law
- SDSS7, level 6 ($\sim 1^\circ$), 15 187 non empty histograms computed in 30s.



Probability of identifications (fast solution)

- Number of spurious match estimates
  - Positional errors sampling for both catalogues
  - $N_{spur} = \sum_A \sum_B S_{conv} / S_{pixel}$
- SDSS7 x 2MASS, level 6, 8min (not yet multithreaded!)