ExXmatch: a tool for extremely fast cross-matches

F.-X. Pineau[1]

[1] CDS, Observatoire Astronomique de Strasbourg

ADASS XXIX - Groningen, 06-10th October, 2019

What is ExXmatch?

- A set of command line tools ...
  - create index, print index information, perform a local cross-match, ...
  - partition index, perform distributed cross-match (master + slave daemon)
- ... to perform xmatches as fast as possible (*efficiency*)
  - computational part x30-60 faster than the current CDS XMatch service!
- ... with a small dedicated memory footprint
- ... running from a laptop to a small cluster of machines (*scalability*).

```
[pineau@cdsxmatch2015 rust-indexfile]$ time ./xmatch_v2 gaia_dr1 sdss8 allsky nn --dmax 3.0 count
Xmatch progress:                                                                    100%
99201152 match found in 4 seconds
99201152

real    0m4.352s
user    1m47.487s
sys     0m15.443s
```

Fig

Example of simple local Gaia DR1 vs SDSS DR8 cross-match (nearest neighbour in 3 arcsec)

Why ExXmatch?

Replace and improve the current CDS XMatch Service

- Get ready for the future: 37 G objects in LSST DR11
- Increase the single machine xmatch performances
- Benefit from a possible cluster of machines
- Run on costs constrained hardware (both size and nbr of machines)
- Energy-efficiency (environmental aspects)
  - try to maximize # associations / thread / cpu clock cycle

```
fxpineau@cds-stg-8:~/rust-indexfile$ RUST_LOG=xmatch_slave=trace ADDRESS="$(hostname -I | tr -d ' '):8080" cargo run
 --release --bin xmatch_slave
warning: unused manifest key: build
    Finished release [optimized] target(s) in 0.04s
     Running `target/release/xmatch_slave`
 INFO  xmatch_slave > XMatch Slave - v0.1.0
 DEBUG xmatch_slave > Starting...
 WARN  xmatch_slave > Can't read config file: No such file or directory (os error 2)
 DEBUG xmatch_slave > Trying to bind server to address: 10.30.2.103:8080
 DEBUG xmatch_slave > Server is listening...
```

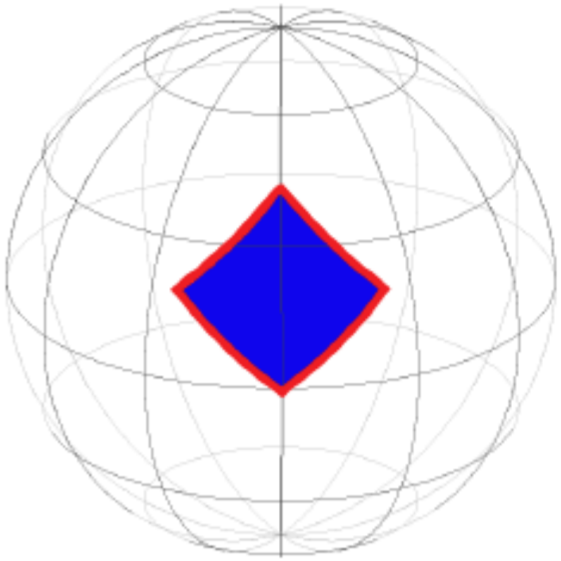
Fig

ExXmatch daemon running on a regular desktop

ExXmatch how?

- Make the cross-match perfectly parallel
  - Use HEALPix tesselation
  - Duplicate data in HEALPix cell borders
  - $\Rightarrow$ each HEALPix cell is independant
- Compress data while allowing random access to individual cells
- Favor non-concurrent sequential reads
- Pre-compute efficient search data structures (kd-tree)
- Ban trignometric functions, square roots, ...
- Resort to multi-threading
- Balance data accross slaves
- Use a performant while safe and memory sober language (Rust)

ExXmatch "bucketing"



- Bucket (in our case):
  - elementary data unit that can be xmatched independently
  - one bucket = one HEALPix cell
- Bucket content:
  - list of sources inside the HEALPix cell
  - + copy of the sources in the cell neighbourhood

- Choices made: compromise between cell size and duplication factor
  - bucket = HEALPix cell at order 8
    - ★ full-sky catalogue $\rightsquigarrow$ 786 432 buckets
    - ★ cell size $\approx 13'7 x 13'7$
  - Neighbourhood size: $30''$
  - $\rightsquigarrow$ duplication factor $\approx 7.5\%$
- Remark: a single bucket may contains 0, 1 or million sources
  - how to perform fast searches in a (possibly large) bucket?

kd-tree bench

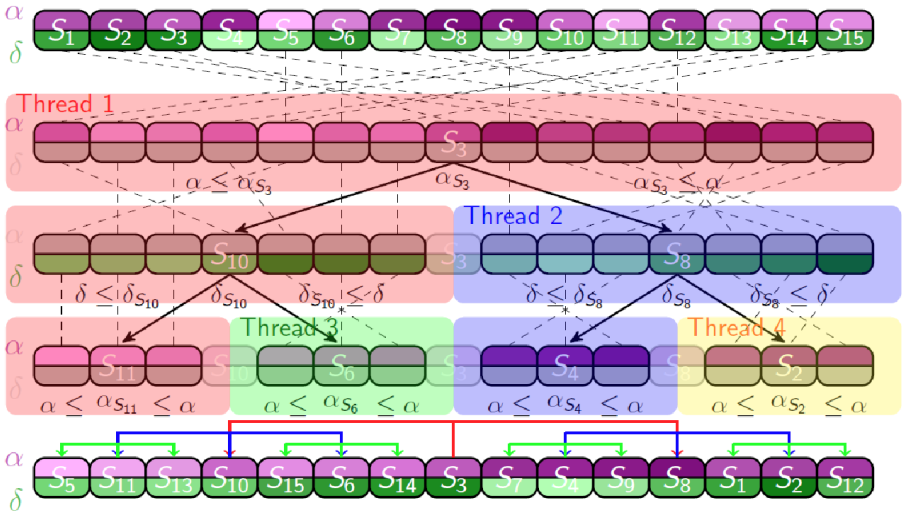
- Solution: 1 bucket = 1 pre-computed kd-tree
- We compared 3 kd-tree algorithms...

  naive implicit *xyz*-tree : pre-defined segregating coordinates: x, y, z, ...

  improved implicit *xyz*-tree : segregating coordinate = the one having the widest range

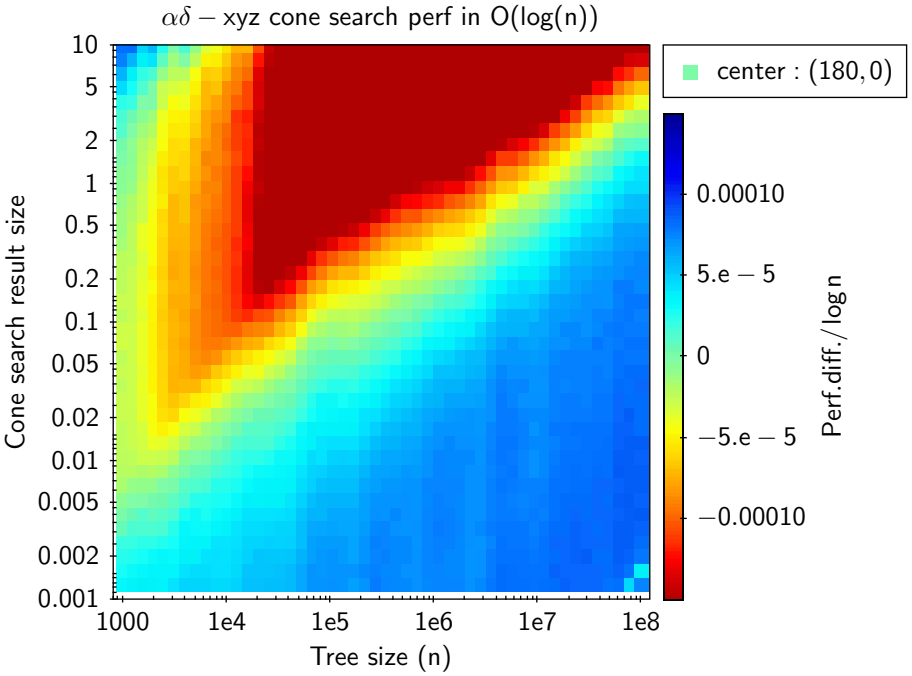
  naive implicit $\alpha\delta$-tree : naive 2d-tree with modified search algorithm.

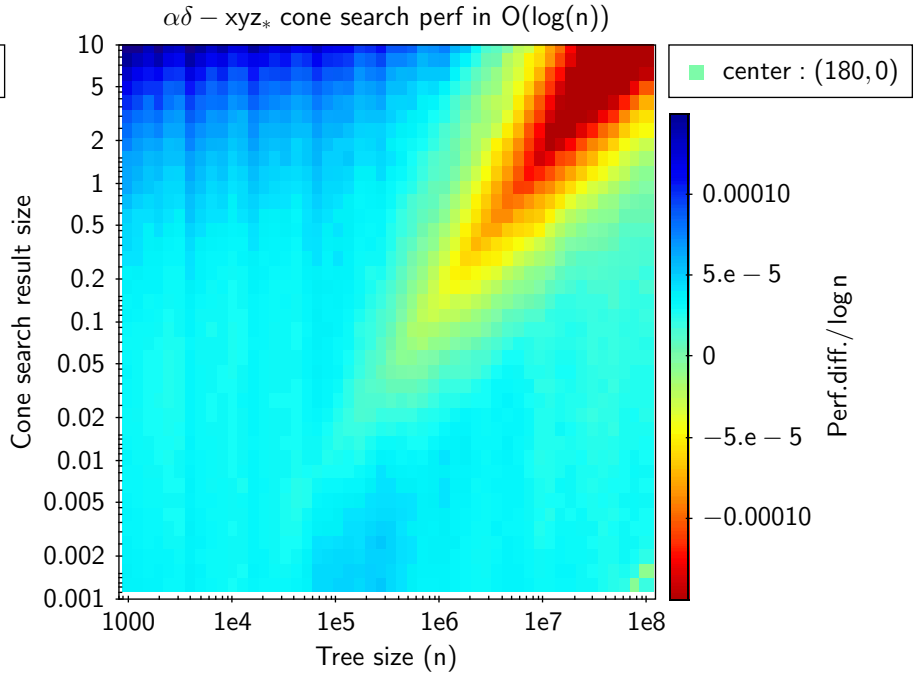
- ... by testing:
  - data centered around various positions:
    - ★ $(\pi, 0)$, $(\pi/4, 0)$, $(\pi/4, \pi 4)$, $(\pi, \pi/2)$
  - various tree sizes: $\in [1e^3, 1e^8]$
  - various average cone search query result size: $\in [0.001, 10]$



kd-tree bench: query time

- Comparing cone search query time around $(\alpha, \delta) = (\pi, 0)$





- Yellow-Red: $\alpha\delta$-tree faster than *xyz* – *tree* or *xyz*$_*$ – *tree*!
  - the *x* axis is not a discriminent axis in this particular case
- Results very different depending on the location on the sphere!

kd-tree: design choice

For best performances and low storage:

- 1 bucket = 1 HEALPix cell = 1 naive implicit *yz*-tree
- *yz*: orthographics coordinates
- projection center: center of the HEALPix cell
- compression possible
  - y and z are offsets (32 bits for each are enough)
  - no need to store x: $x = \sqrt{1 - (y^2 + z^2)}$
- Searches in a 2D-tree with distances computed in 3D

Fast yet precise angular distance

- From orthographic (y, z) coordinates < 30 arcmin and distances < 30 arcsec:

$$\rho^2 = \Delta^2 y + \Delta^2 z + \left(\frac{\Delta y (y_B + y_A) + \Delta z (z_B + z_A)}{2} \right)^2 \tag{1}$$

- Precision of $\approx 1\mu arcsec$
- Only fast CPU operations: ~~sqrt, trigonometric functions~~
- Tested on SDSS DR8 vs 2MASS xmatch at 5 arcsec
  - Same number of match: 66 018 041
  - Arcsin : `real 0m5.704s; user 2m42.350s; sys: 0m13.761s`
  - Approx : `real 0m3.917s; user 1m38.747s; sys: 0m12.832s`
  - $\Rightarrow$ 30% performance improvement (40% less CPU time!)

ExXmatch index file

- An index is made of 2 files:
  - A metadata file (TOML format): #entries, datatypes, COOSYS, ...
  - The index data file (binary format)
    - ★ implicit HPX indices: 2 values per bucket
    - ★ implicit kd-trees (sorted arrays)

Data file

index part
data part

Index part

bucket 0: nSrcs (cumul)
bucket 1: nSrcs (cumul)
...
bucket n: nSrcs (cumul)
bucket 0: nSrcs in border
bucket 1: nSrcs in border
...
bucket n: nSrcs in border

Data part

Src 0: id,y,z	b	k
Src 1: id,y,z	u	d
Src 2: id,y,z	c	t
Src 3: id,y,z	k	r
Src 4: id,y,z	e	e
Src 5: id,y,z	t	e
Src 6: id,y,z		
Src 7: id,y,z		
...		
Src n: id,y,z		

Cross-match timeline

- Each number corresponds to a *chunk* of data
- A *chunk* is an ordered group of buckets (HEALPix cells) to be processed

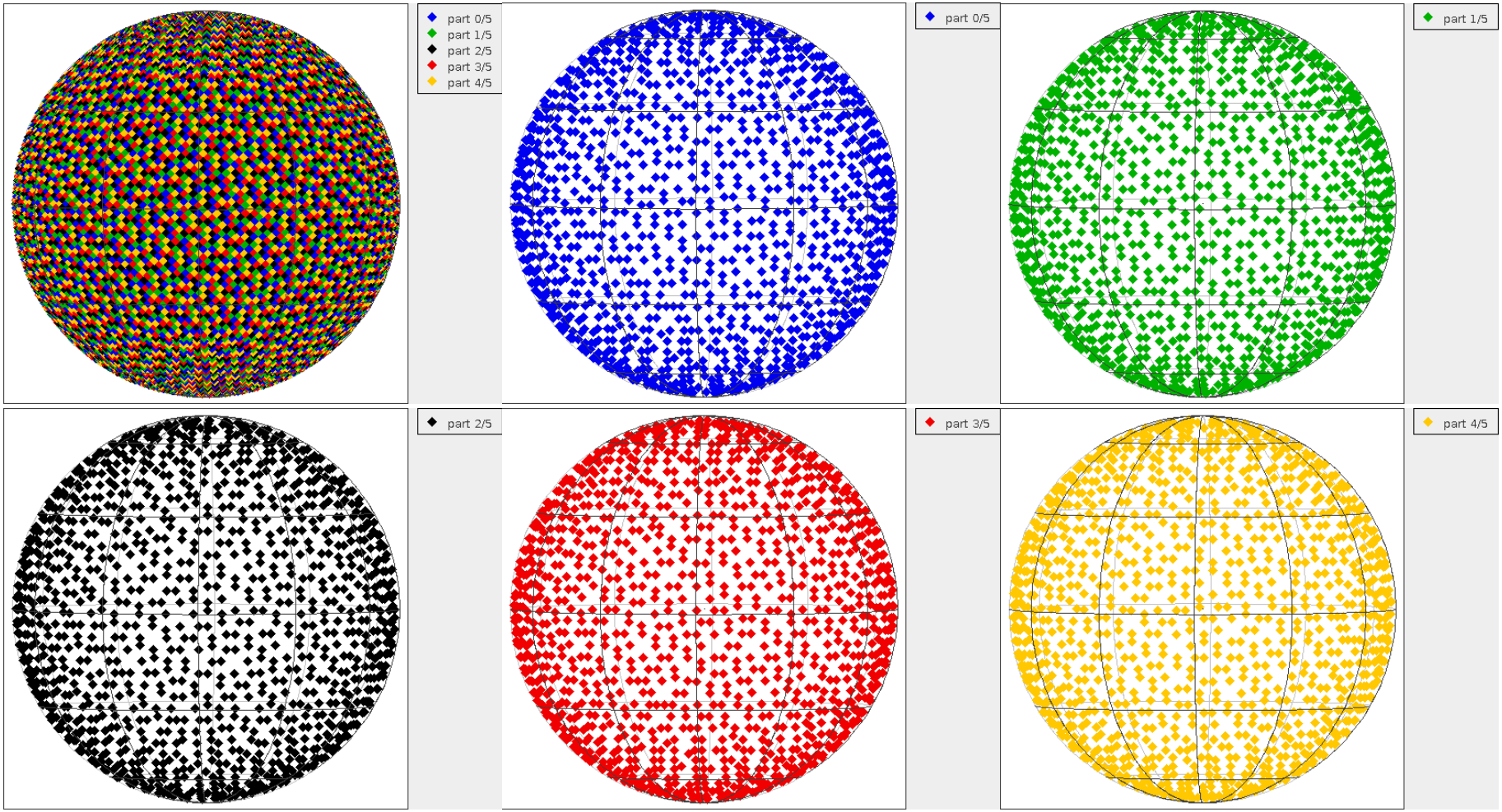
Op. \Chunk index		Chunk 0	Chunk 1	Chunk 2	...
Read file A	-0-	-1-	-2-	-3-	...
Read file B	-0-	-1-	-2-	-3-	...
Compute matches		----0----	----1----	--2--	...
Write result file			-0-	-1-	...
Limiting factor		CPU		I/O	

- Both the read operations and the cross-match are multi-threaded
- Loading a *chunk* of data while processing the previous chunk
  - improves performances with a *cold* cache;
  - does not modify performances with a *warm* cache.

Distributed xmatch

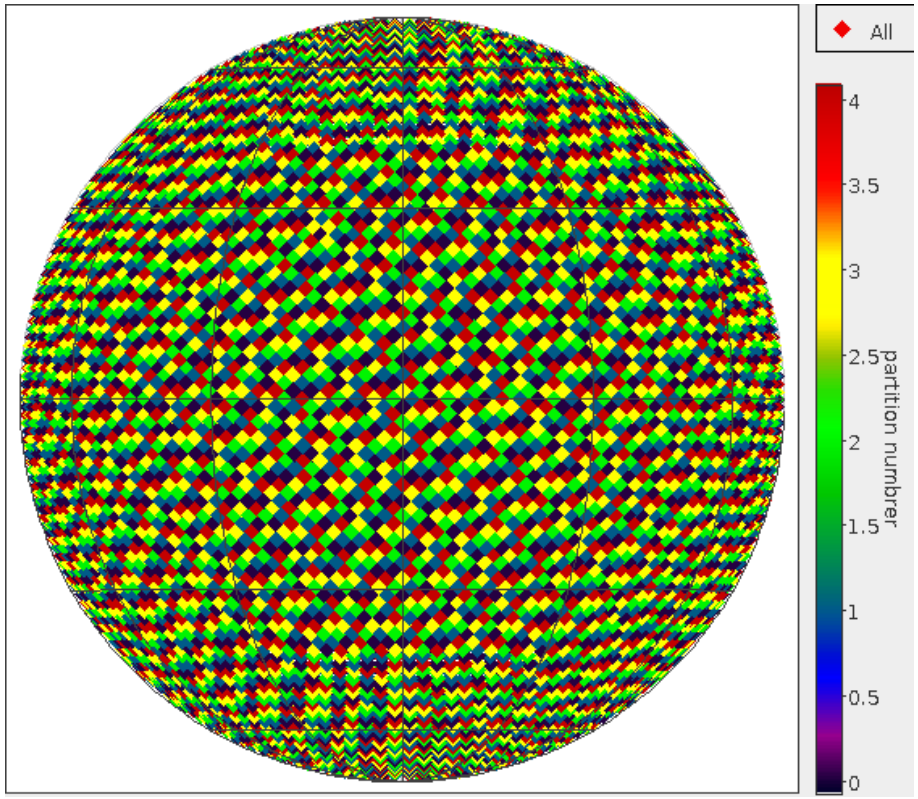
Use case: cluster of 5 identical machines

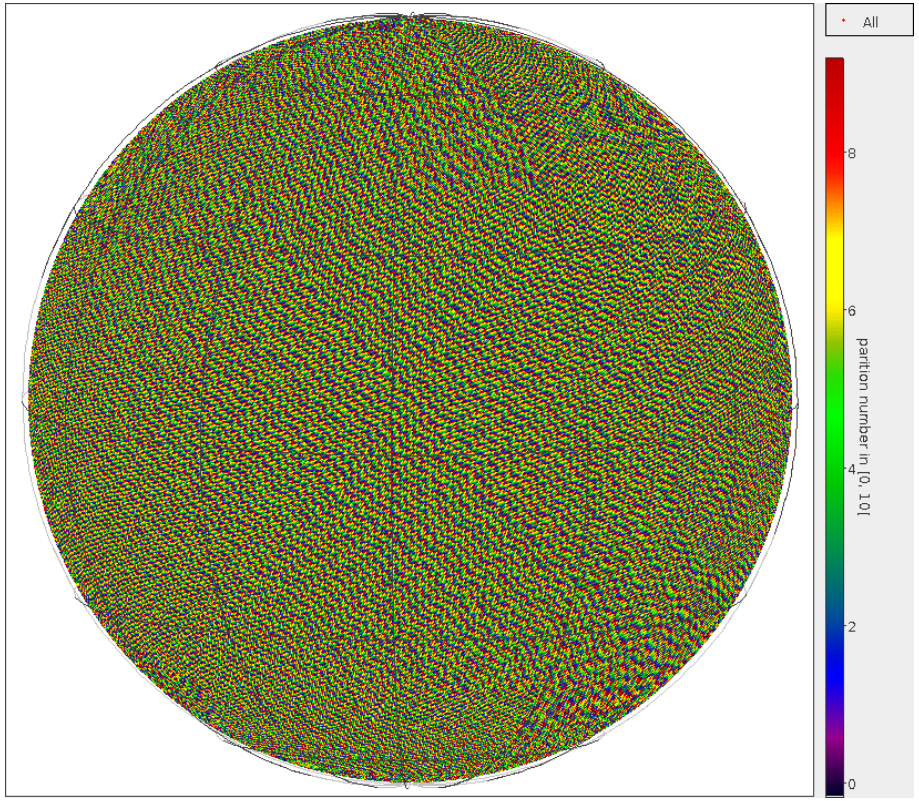
- 5 partitions: 1 partition per slave
- Cell number i in partition number $p = i \mod 5$



Distributed xmatch

- Left: 5 partitions at an HEALPix order 5 (illustrative)
- Right: 10 partitions at an HEALPix order 8 (order actually used)

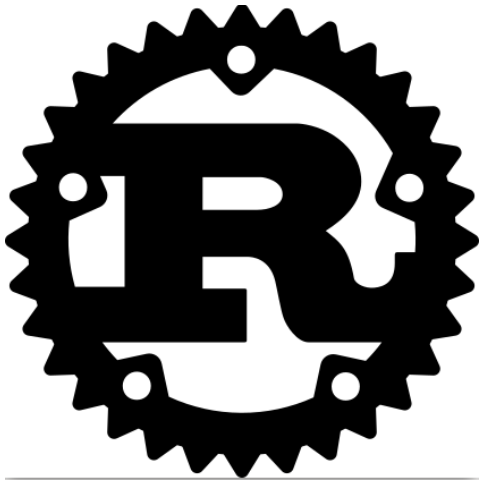


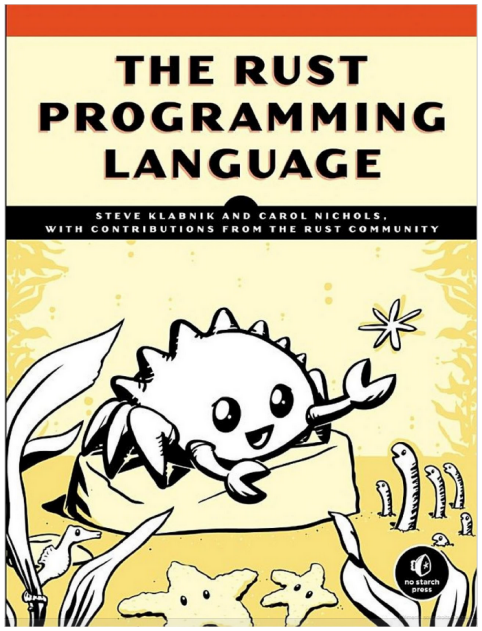


- ⇒ very well balance distribution of data and CPU time over all slaves (as presented at IVOA in Triestre)!

ExXmatch language: RUST!

- Why RUST?
  - As fast as C, C++
  - But safer: no seg. fault, buffer overflow, ...
  - Compiled: no startup overhead
  - No GC: predictable performances, lower memory consumption
  - Easy to embed in orther popular languages (Python, Javascript/WebAssembly, ...)
  - Fearless concurrency
  - Modern environement: Cargo, crates.io, markdown doc, ...
  - Modern and multiparadigm: type inference, closures, functional aspects, async, ...
  - Second most energy-efficient language (C is first)
  - StackOverflow most loved language in 2016, 2017, 2018 and 2019
  - Used in Facebook Libra, Firefox, Dropbox, ...





Benchmark: hardware

Machines:

- desktop computer
  - Intel core i5-7500 (4 cores, 4 threads, 3.4-3.8 GHz)
  - 16 GB RAM 2400 MHz
  - NVMe SSD
- server:
  - 2 x Intel Xeon E5-2650 v3 (2x10 cores, 2x20 threads, 2.3-3.0 GHz)
  - 64 GB RAM 2133 MHz
  - 5x SATA SSDs Samsung EVO
- cluster (machines in 4 different places)
  - 1 server
  - 7 desktop computers
  - 9 partitions: 1 per desktop + 2 on the server

Benchmark results

- Compute and count the number of matches

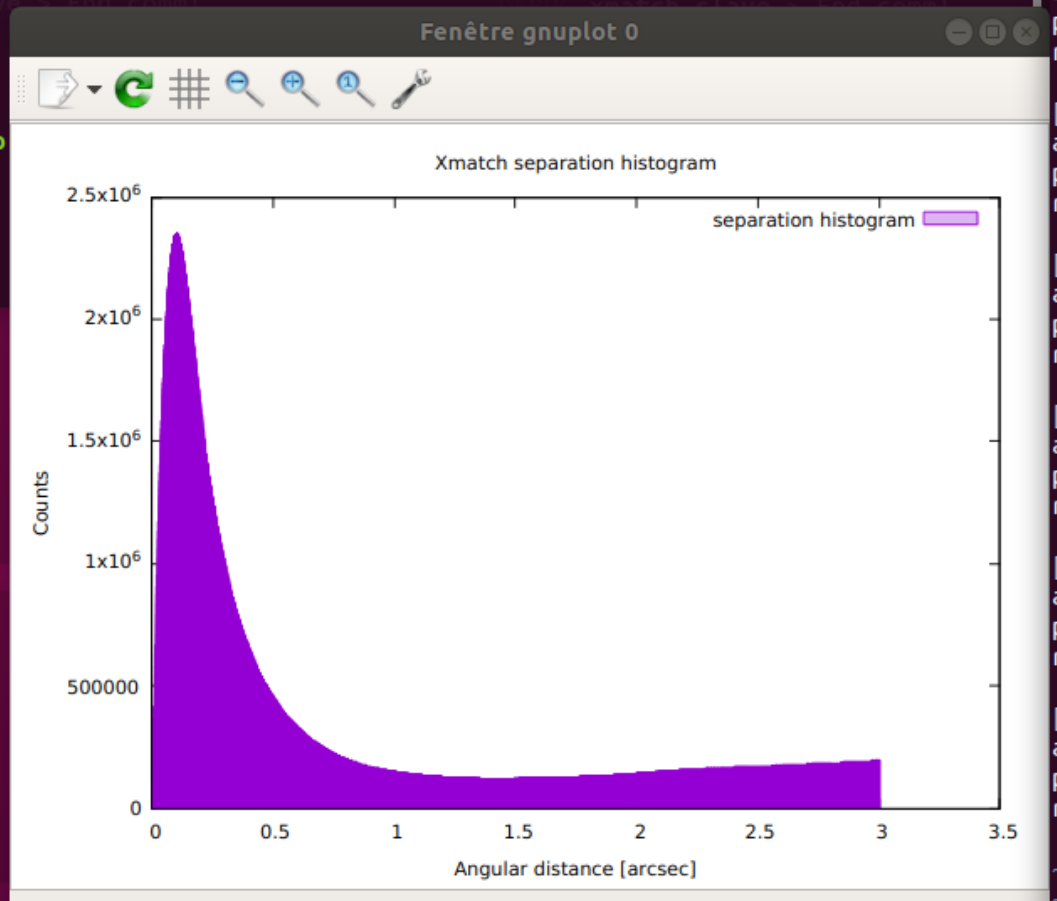
L table	R table	Params	nLinks	cache	desk.	serv.	clust.
2MASS	SDSS DR8	nn in 5"	54 M	cold	8.7 s	5.1 s	1.2 s
2MASS	SDSS DR8	nn in 5"	54 M	warm	7.8 s	3.5 s	0.9 s
GAIA DR1	SDSS DR8	cone 5"	125 M	cold	14.2 s	6.4 s	1.8 s
GAIA DR1	SDSS DR8	cone 5"	125 M	warm	13.0 s	4.9 s	1.6 s
2MASS	GAIA DR1	2nn in 5"	577 M	cold	49.8 s	18.9 s	5.8 s
2MASS	GAIA DR1	2nn in 5"	577 M	warm	49.8 s	16.3 s	5.6 s
GAIA DR1	IGSL3	nn in 5"	947 M	warm	69.5 s	20.0 s	8.4 s
GAIA DR1	IGSL3	cone 5"	1.6 G	cold	100.4 s	33.2 s	11.3 s
GAIA DR1	IGSL3	cone 5"	1.6 G	warm	100.4 s	29.2 s	11.0 s

- Number of sources in
  - SDSS DR8 (primary sources): 469 M;
  - 2MASS: 470 M; Gaia DR1: 1.14 G; IGSL: 1.22 G
- Drop HDD cache: `sync; echo 3 > /proc/sys/vm/drop_caches`
- Cold/warm cache: same perfs when data size larger than available RAM.

Benchmark: histogram

```
pineau@cds-dev-fxp:~/IdeaProjects/rust-indexfile$ export step=0.001
pineau@cds-dev-fxp:~/IdeaProjects/rust-indexfile$ time RUST_LOG=xmatch_master=error .
/xmatch_master -c config/slaves.toml gaia_dr1 igsl3 allsky dist -r 3.0 histo -s ${ste
p} | ./gp_histo
[00:00:10] ######################################## 100% 10.30.2.96:8080 - success
[00:00:11] ######################################## 100% 10.30.2.98:8080 - success
[00:00:10] ######################################## 100% 10.30.2.99:8080 - success
[00:00:10] ######################################## 100% 10.30.2.100:8080 - success
[00:00:11] ######################################## 100% 10.30.2.101:8080 - success
[00:00:11] ######################################## 100% 10.30.2.102:8080 - success
[00:00:11] ######################################## 100% 10.30.2.103:8080 - success
[00:00:10] ######################################## 100% 10.30.6.25:8080 - success
Completed: 8; Failed: 0
Histogram contains 1058702383 matches.

real    0m11,452s
user    0m0,103s
sys     0m0,087s
pineau@cds-dev-fxp
```




```
n_partition = 9

[[slaves]]
addr = "10.30.2.96:8080"
partitions = [0]
replicas = []

[[slaves]]
addr = "10.30.2.98:8080"
partitions = [1]
replicas = []

[[slaves]]
addr = "10.30.2.99:8080"
partitions = [2]
replicas = []

[[slaves]]
addr = "10.30.2.100:8080"
partitions = [3]
replicas = []

[[slaves]]
addr = "10.30.2.101:8080"
partitions = [4]
replicas = []

[[slaves]]
addr = "10.30.2.102:8080"
partitions = [5]
replicas = []

[[slaves]]
addr = "10.30.2.103:8080"
partitions = [6]
replicas = []

[[slaves]]
addr = "10.30.6.25:8080"
partitions = [7, 8]
replicas = []

~
```

Future plans

- Simple cross-match computation time: not a problem
- Problem: output file size containing the catalogue columns ($> 1\,\mathrm{TB}$)
  - improve serialisation/deserialisation performance?
    - try to reach $1\,\mathrm{GB/s}$?
  - allow users to choose the columns they want
  - allow users to pre and post-filter catalogues
  - allow users to make selections and plots on the server side?
- Support more complex xmatch algorithm (ARCHES!)
  - take into account positional errors
  - take into account proper motions
  - ...