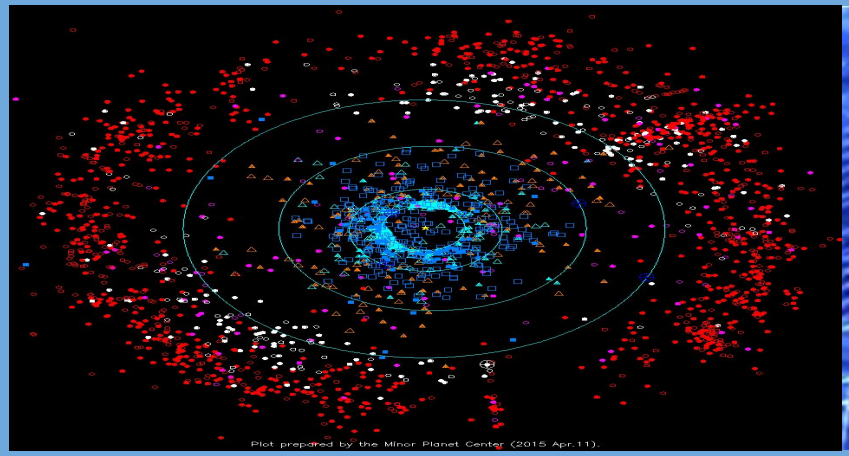
Parallelizing Identification of Slow Moving Objects in the Outer Solar System Pipeline



CS 205: Advanced Scientific Computing

Jon Clindaniel, Michael Lackner, Karen Yu + Matt Holman, CfA

High-performance computing is crucial for identifying objects in the outer solar system

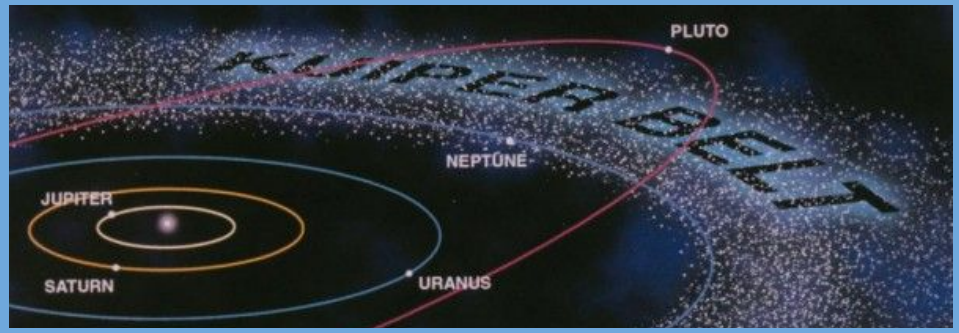


Diagram illustrating the Kuiper Belt, which includes Pluto and two other minor planets (http://theplanets.org/kuiper-belt/)

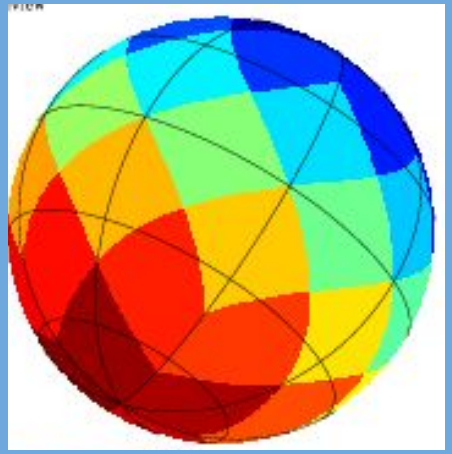
Above: Pan-STARRS telescope located at the summit of Haleakala, Maui, Hawaii
Right: Shoemaker Levy-9 colliding with Jupiter (http://www2.jpl.nasa.gov/sl9/)

From Pan-STARRS to the LSST

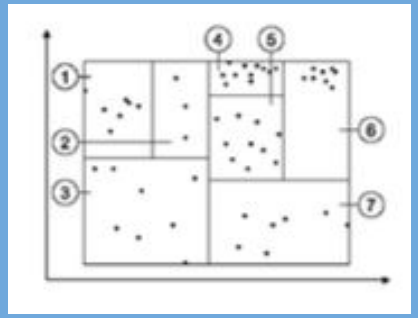
Pan-STARRS (2015)	→	**LSST (approx 2022)**
Discrete exposures / night	→	“Movies of the sky” 2 terabyte/hr stream
2 PETABYTES (4 yrs)	→	150 PETABYTES (10 yrs)
Object ID off-line	→	Object ID real time

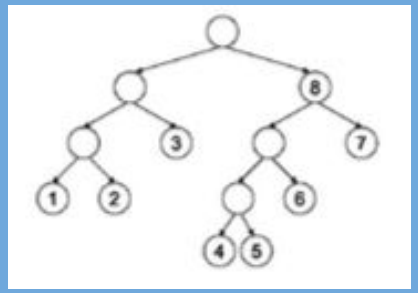
The future of Time-Domain Astronomy depends on parallelization

Current serial process identifies slow-moving objects by fitting detections to orbital “tracklets”

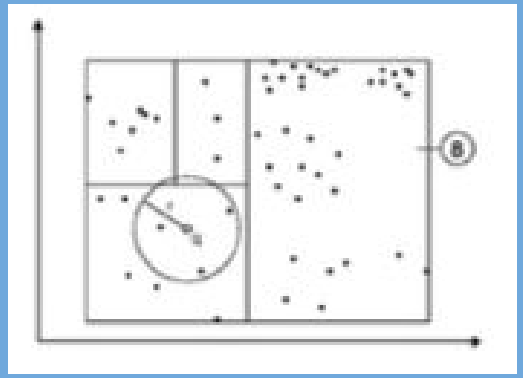


Sky is divided into 9000 tiles. Each tile has 4 years of multiple nightly exposures. Terabytes of exposure/detection data.





Construct kd-tree of detections.

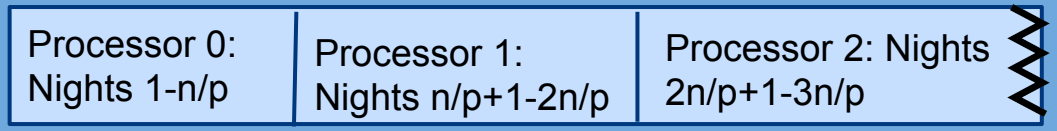


Query tree to construct tracklets.

Diagrams from Kubica et al. 2007.

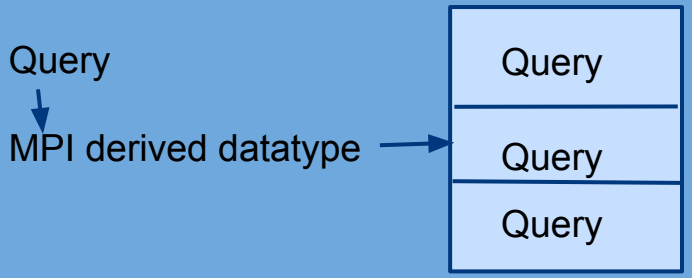
Our Approach: An MPI approach to tracklet linking

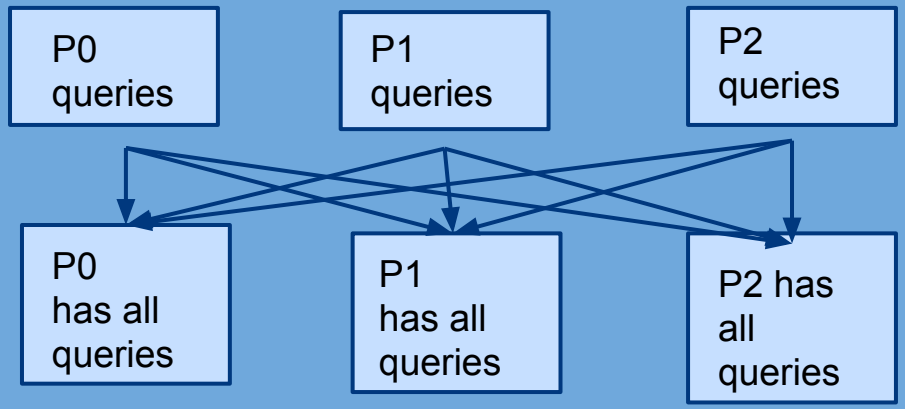
1. Temporal decomposition: divide nights among processors:



2. Each processor computes tracklets for the nights it is in charge of.

3. Information needed from other processors is bundled into a Query vector.





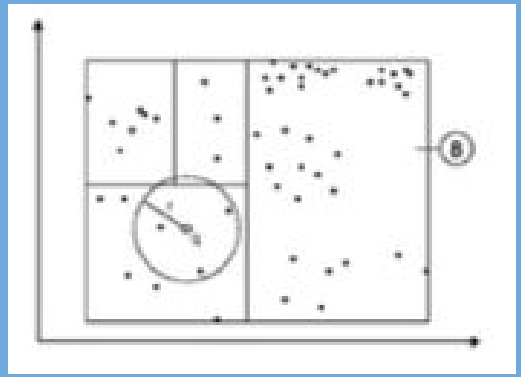
4. Queries are sent to all processors using allgatherv

C++ / MPI code development

```
int global_NumQueries = 0; // total number of queries for all processors
int NumQueries = queries.size(); // number of queries for this processor

// send query totals to allgather
int rCounts[nprocs]; // hold each processor's subtotal of overall query total
MPI::COMM_WORLD.Allgather(&NumQueries, 1, MPI::INT, &rCounts, 1, MPI::INT);

int rDisp[nprocs]; // displacement from receive buffer for each processor's queries
rDisp[0] = 0; // processor 0's displacement is always 0;

/* calculate displacements */
for (int i = 0; i < nprocs; ++i) {
    global_NumQueries += rCounts[i]; // total queries all processors
    if (i != 0) {
        rDisp[i] = rDisp[i - 1] + rCounts[i - 1];
    }
}

printf("Number of queries for rank %d =  %d  : total queries all ranks = %d : total detections %d : total tracklets %d\n",
        rank, NumQueries, global_NumQueries, ossp.num_nodes(), ossp.size());
fflush(stdout);
Query rcvQueries[global_NumQueries]; // array to receive queries from all processors
// queries[0] below has the effect of exposing the array within the vector 'queries']
MPI::COMM_WORLD.Allgatherv(&queries[0], rCounts[rank], MPIQuery, &rcvQueries, rCounts, rDisp, MPIQuery);

/* unpack rcvqueries array into queries vector*/
queries.clear();
queries.insert(queries.end(), &rcvQueries[0], &rcvQueries[global_NumQueries]);
```

Our Approach: An MPI approach to tracklet linking

5. Each processor evaluates all queries. Sends detection information it has to processors with linkable tracklets.



6. Each processor uses remote detections to create local tracklets.

Overall Results

- Example: 38 seconds → 9 seconds using 4 processors
- O(N) → O(1) improvement using N processors
- Output (i.e. # and detection count of tracklets) identical to serial process
- Functional requirements met
- TODO: implement optimistic concurrency using RDBMS and enforce consistency

Bottom Line:

- Met performance goal: successfully sped up an existing serial algorithm
- The coming LSST data deluge requires parallelizing many Time-Domain Astronomy algorithms
- In general, re-engineering existing serial process infrastructure for parallel operations is increasingly important in all domains
- Our “secret weapon” - a domain expert who “speaks code”