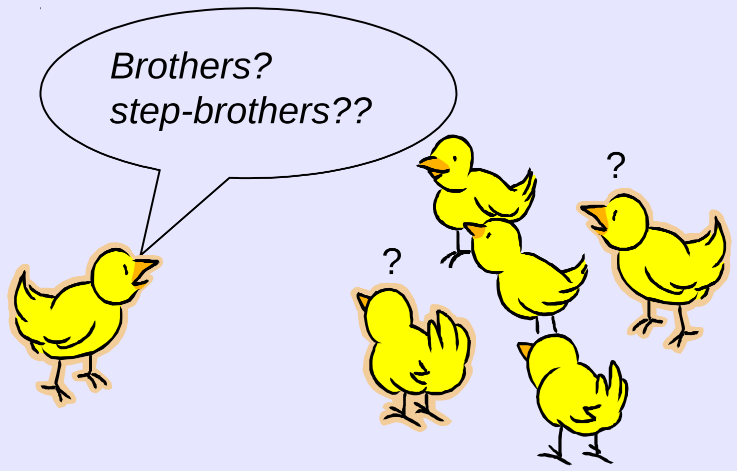
Parallel, Error-Tolerant Sibling Reconstruction



Presented by:
Alan Perez-Rathke

Outline

- **Motivation and Background**
- Improvements to Parallel 2-Allele
- Parallel 2-Allele Consensus
- Future Work
- Q & A

Biological Motivation

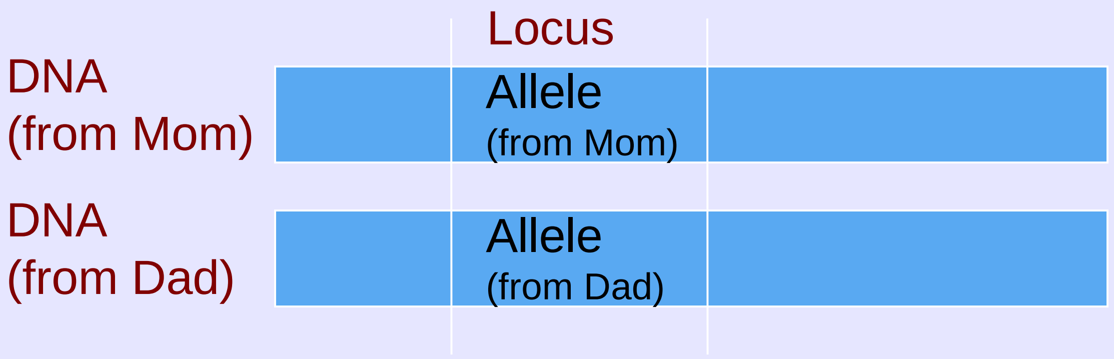
- **Used in**: conservation biology, animal management, molecular ecology, genetic epidemiology
- **Necessary for**: estimating heritability of quantitative characters, characterizing mating systems and fitness.
- **Caveat**: hard to sample parent/offspring pairs. Sampling cohorts of juveniles is much easier.

Lemon sharks, *Negaprion brevirostris*

2 Brown-headed cowbird (*Molothrus ater*) eggs in a Blue-winged Warbler's nest

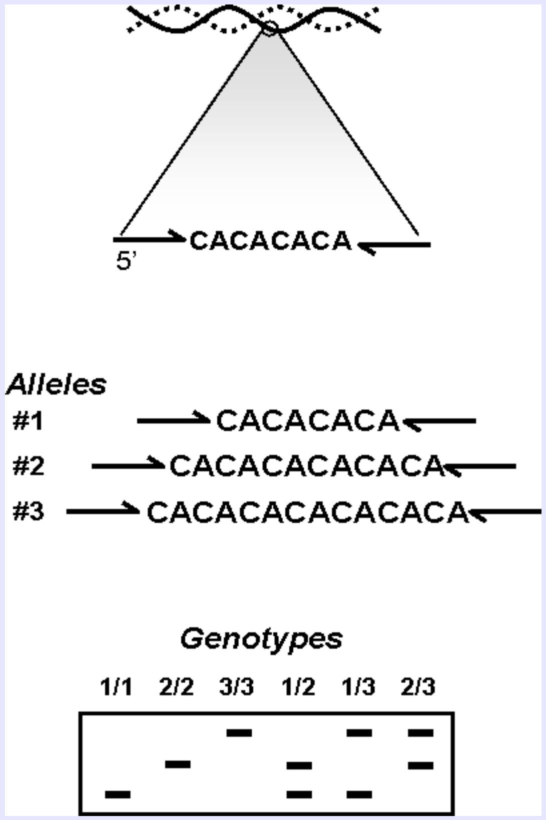
Basic Genetics

- Gene: Unit of inheritance
- Allele: Actual genetic sequence
- Locus: Location of allele in entire genetic sequence
- Diploid: 2 alleles at each locus

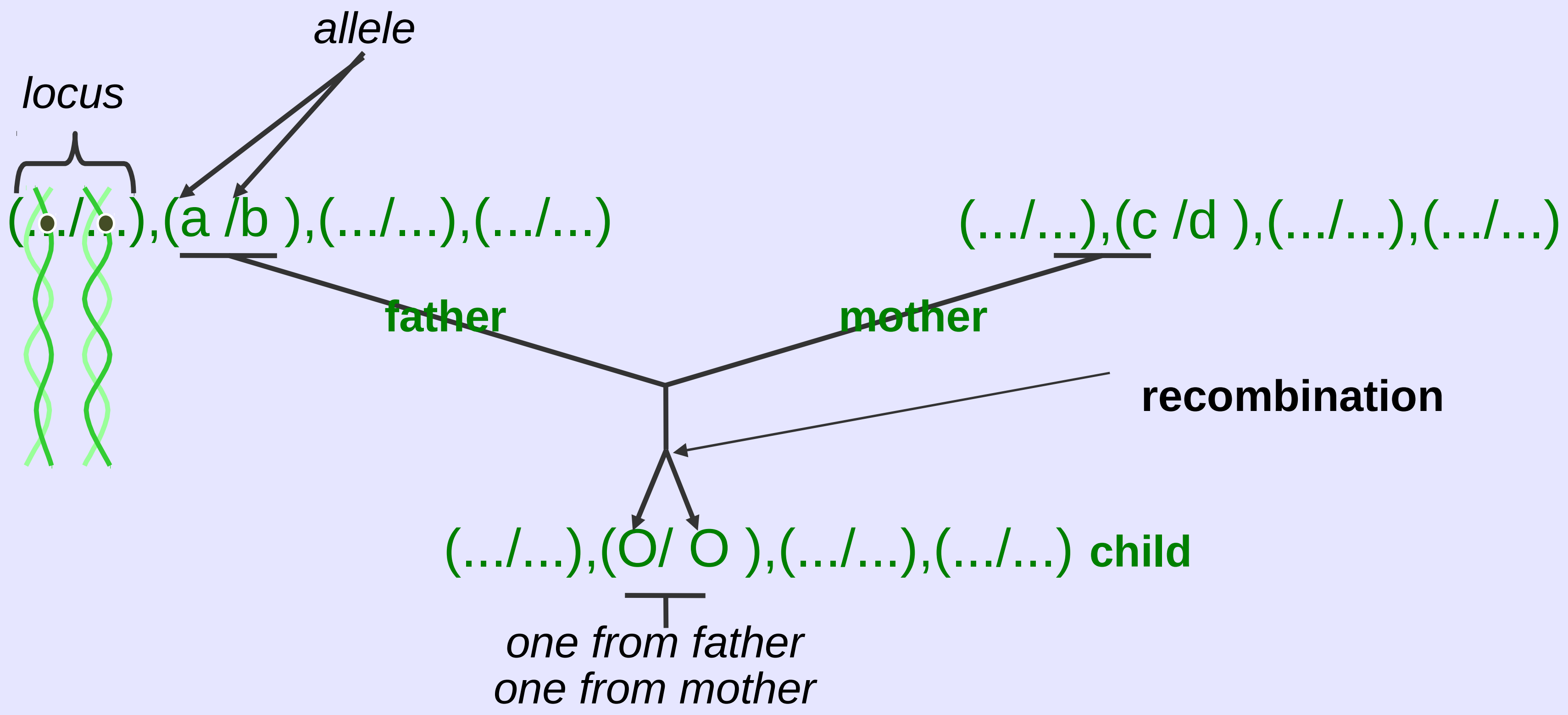


Microsatellites (STR)

- Advantages
  - Equal probability of allele inheritance (easy inference of genotypes and allele frequencies)
  - Many heterozygous alleles per locus
  - Possible to estimate other population parameters
  - Cheaper than SNPs
- But
  - Few loci
- And
  - Large families
  - Self-mating



Diploid Siblings


allele
locus
(.../...),(a /b),(.../...),(.../...)
(.../...),(c /d),(.../...),(.../...)
father
mother
recombination
(.../...),(O/ O),(.../...),(.../...) child
one from father
one from mother

Siblings: Children with the same parent(s)

Question: Given a set of children, can we find the sibling groups?

Sibling Reconstruction Problem

Individual	Locus1	Locus2
	allele1/allele2	
1	111/122	211/222
2	111/133	233/244
3	111/144	233/255
4	111/133	277/266
5	111/133	233/244
6	111/133	233/277
7	111/155	288/222
8	111/166	222/222

$\longrightarrow$

Sibling Groups:

2, 4, 5, 6

1, 3

7, 8

Existing Methods

Method	Approach	Error-Detectio	Assumptions
Almudevar& Field (1999,2003)	Minimal Sibling groups under likelihood	No	Minimal sibgroups, representative allele frequencies
KinGroup (2004)	Markov Chain Monte Carlo/ML	No	Allele Frequencies etc. are representative
Family Finder(2003)	Partition population using	No	Allele Frequencies etc. are representative
Pedigree (2001)	Markov Chain Monte Carlo/ML	No	Allele Frequencies etc are representative
COLONY (2004)	Simulated Annealing under	Yes	Monogamy for one sex
Fernandez & Toro (2006)	Simulated Annealing	No	Co-ancestry matrix is a good measure, parents can be reconstructed or are available

Mendelian Constraints

4-allele rule:

siblings have at most 4 different alleles in a locus

Yes: 3/3, 1/3, 1/5, 1/6

No:3/3, 1/3, 1/5, 1/6, 3/2

2-allele rule:

In a locus in a sibling group:

$$a + R \le 4$$

Num distinct alleles

Num alleles that appear with 3 others or are homozygote

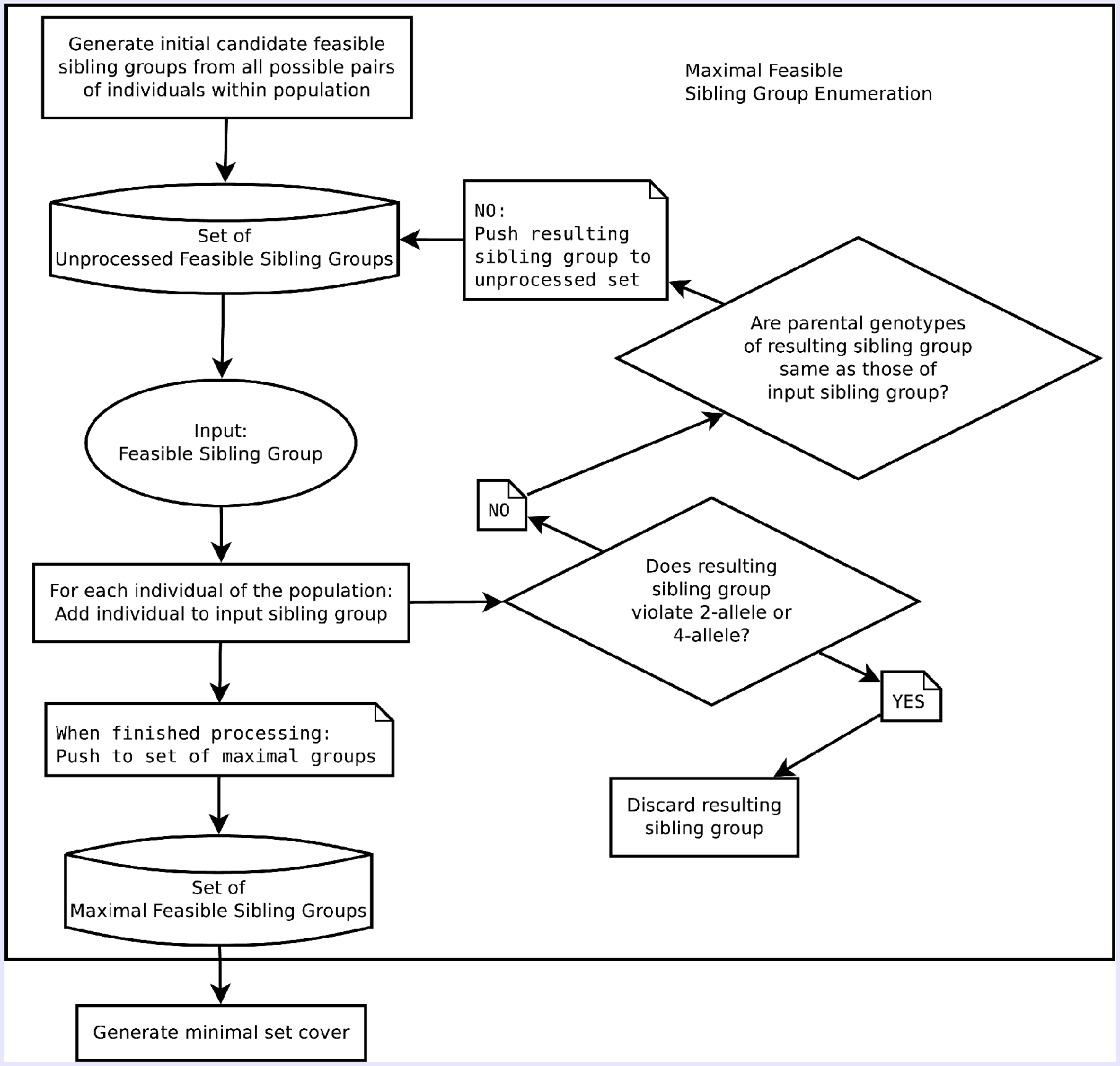
Yes: 3/3, 1/3, 1/5

No: 3/3, 1/3, 1/5, 1/6

2-Allele Min Set Cover

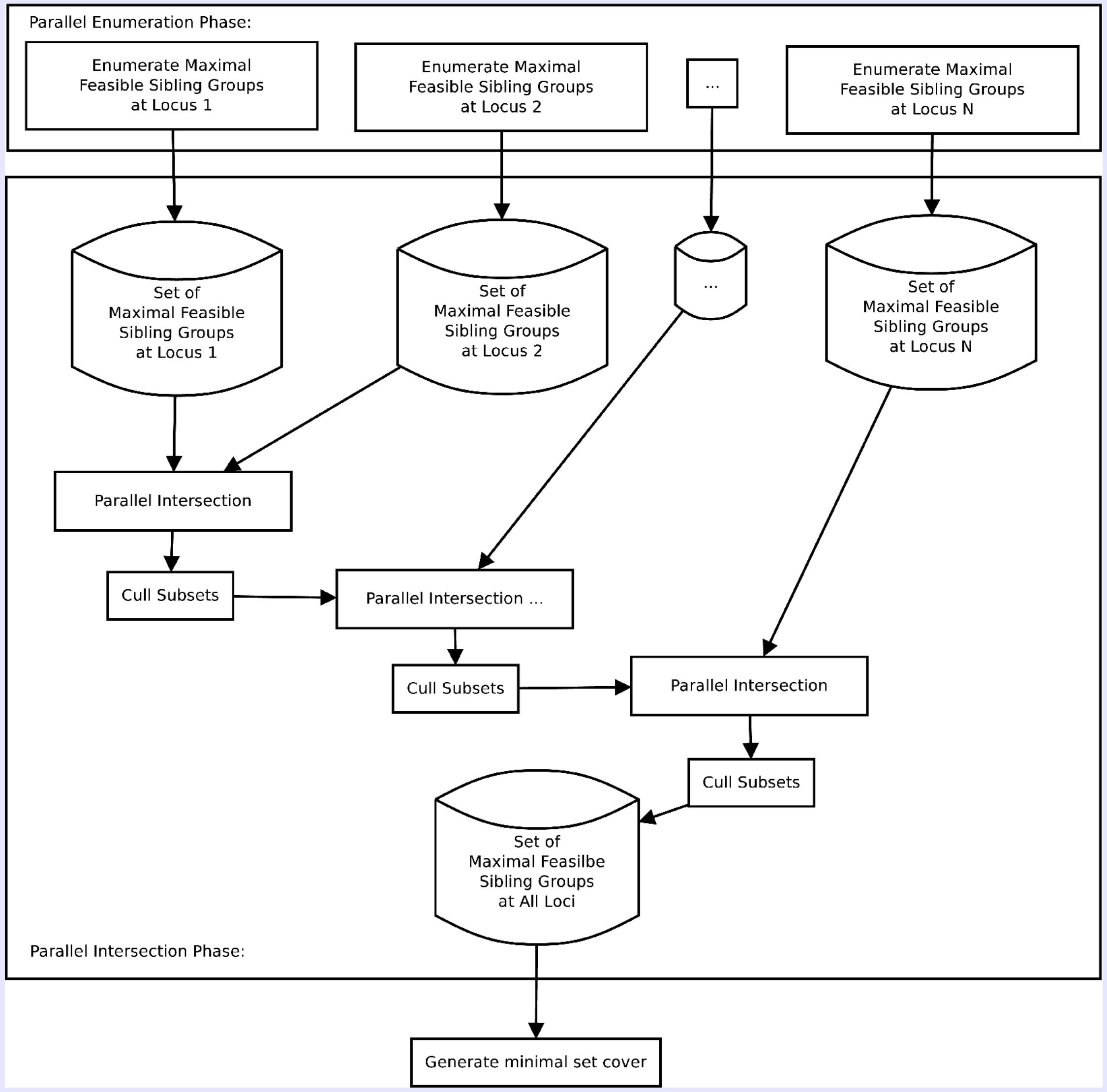
- **Problem Statement**
  - Find the minimum number of sibling groups necessary to explain the given cohort
- **Exact Algorithm** [ISMB/ECCB 2007,Bioinformatics]
  - Enumerate all maximal feasible sibling groups
  - Find min set cover using CPLEX
- **Complexity**
  - NP-Hard & No polynomial approximation [Ashley et al JCSS 09]

Serial 2-Allele Min Set Cover

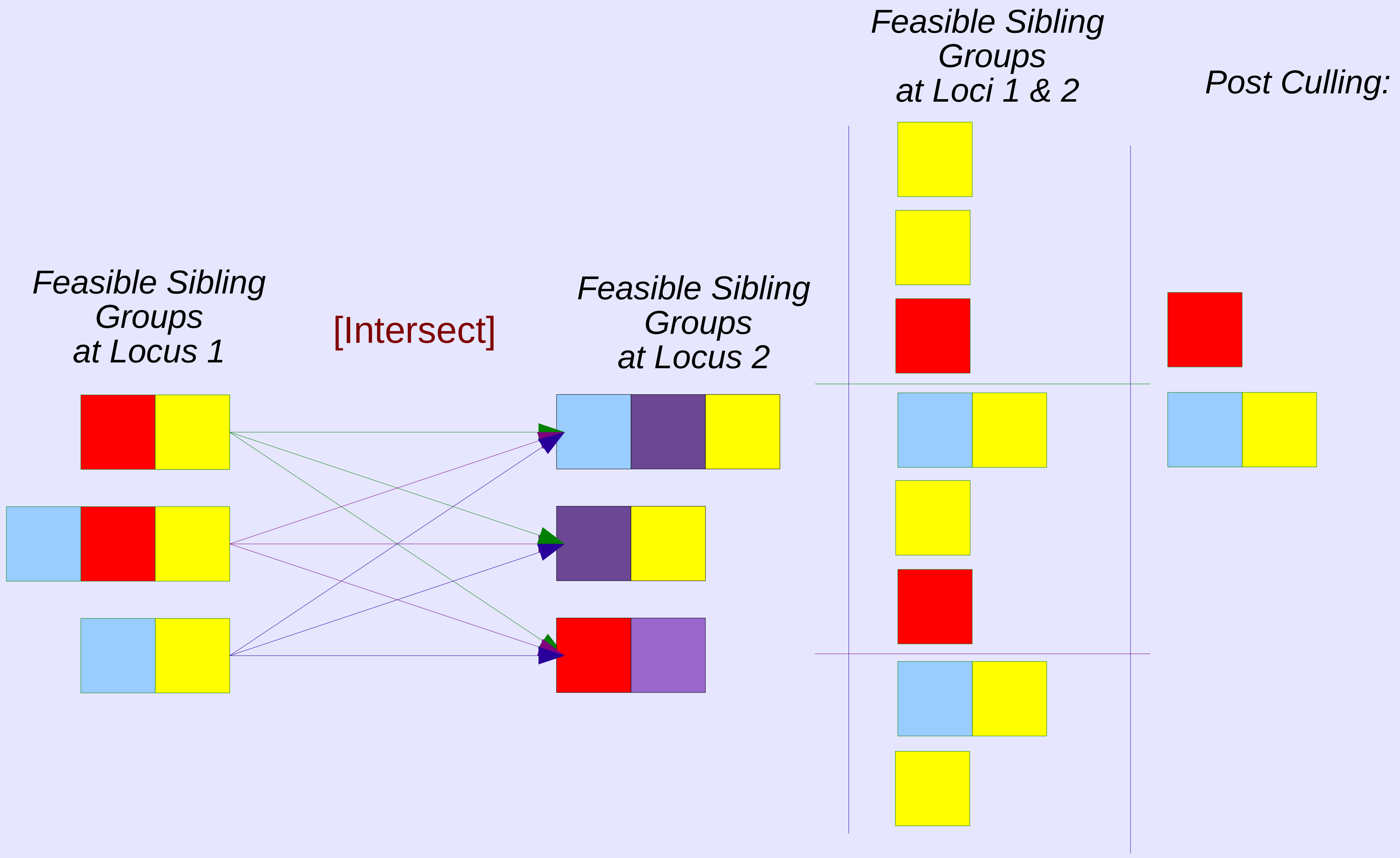

Generate initial candidate feasible sibling groups from all possible pairs of individuals within population
Maximal Feasible Sibling Group Enumeration
Set of Unprocessed Feasible Sibling Groups
NO: Push resulting sibling group to unprocessed set
Are parental genotypes of resulting sibling group same as those of input sibling group?
Input: Feasible Sibling Group
NO
Does resulting sibling group violate 2-allele or 4-allele?
For each individual of the population: Add individual to input sibling group
YES
When finished processing: Push to set of maximal groups
Discard resulting sibling group
Set of Maximal Feasible Sibling Groups
Generate minimal set cover

Parallel 2-Allele Min Set Cover

- Enumeration is slow, how to speed it up?
- Enumerate at each locus in parallel
- Sibgroups are represented as bitsets
- Intersect groups at each locus for final enumeration
- Subsets must be culled


Parallel Enumeration Phase:
Enumerate Maximal Feasible Sibling Groups at Locus 1
Enumerate Maximal Feasible Sibling Groups at Locus 2
...
Enumerate Maximal Feasible Sibling Groups at Locus N
Set of Maximal Feasible Sibling Groups at Locus 1
Set of Maximal Feasible Sibling Groups at Locus 2
...
Set of Maximal Feasible Sibling Groups at Locus N
Parallel Intersection
Cull Subsets
Parallel Intersection ...
Cull Subsets
Parallel Intersection
Cull Subsets
Set of Maximal Feasilbe Sibling Groups at All Loci
Parallel Intersection Phase:
Generate minimal set cover

Example: Intersecting Loci Sets

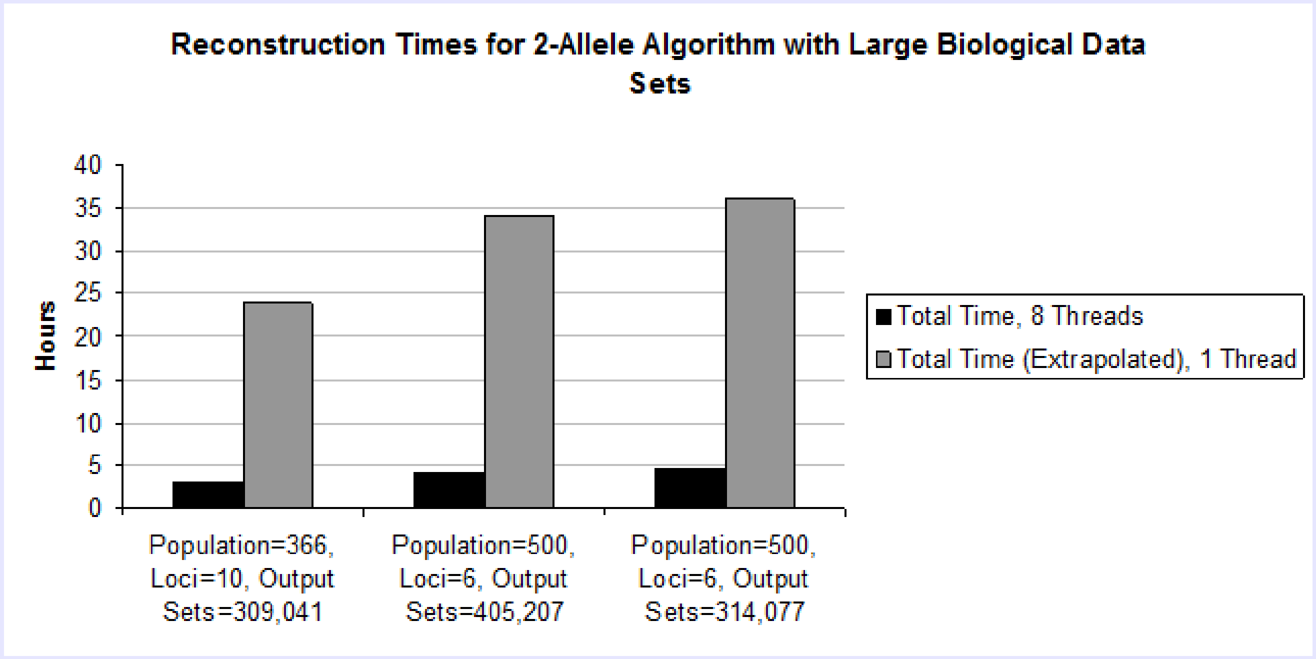

Feasible Sibling Groups at Locus 1
[Intersect]
Feasible Sibling Groups at Locus 2
Feasible Sibling Groups at Loci 1 & 2
Post Culling:

Outline

- Motivation and Background
- **Improvements to Parallel 2-Allele**
- Parallel 2-Allele Consensus
- Future Work
- Q & A

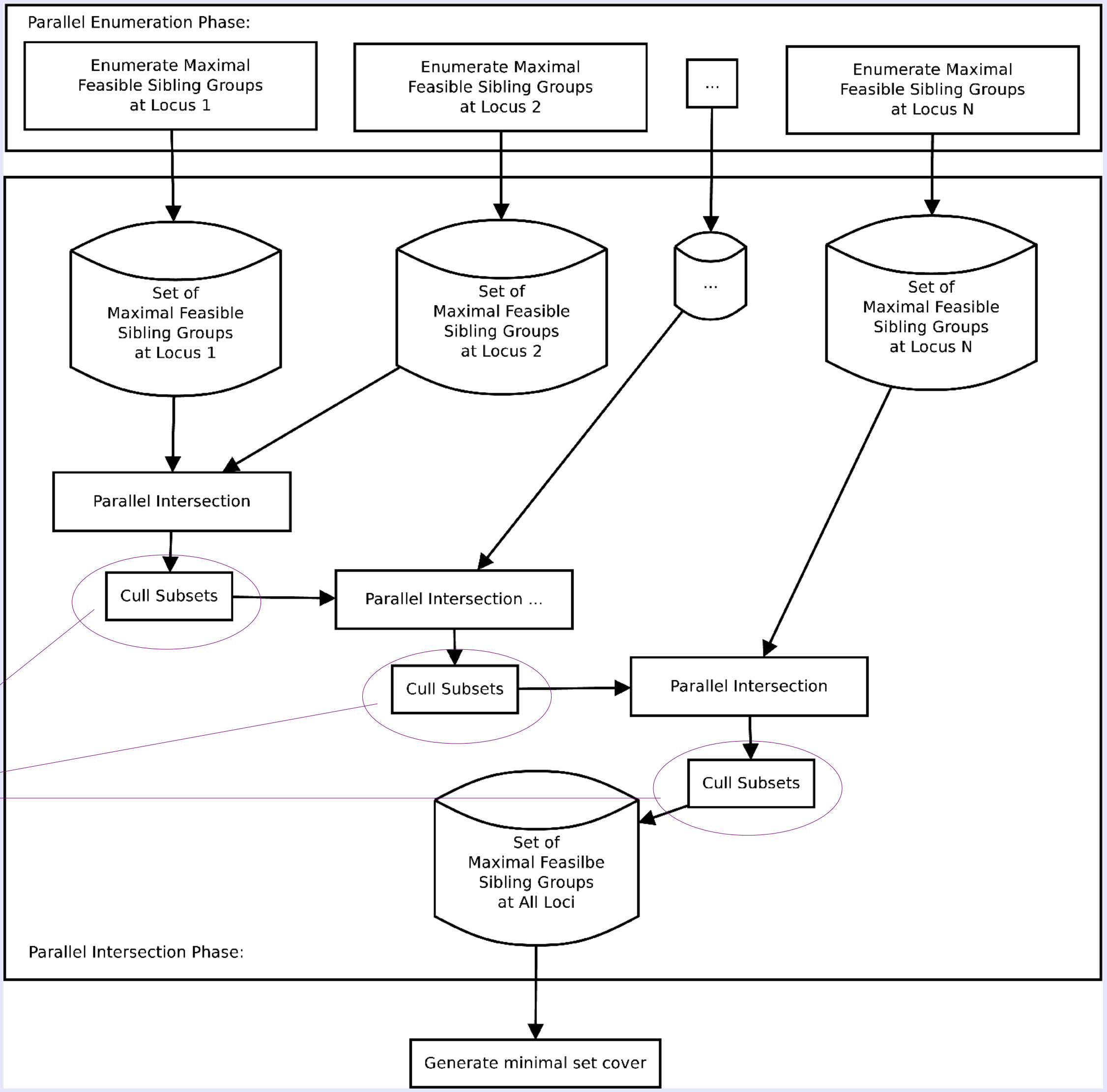
Why Parallel?

- Min Set Cover is NP-hard and no polynomial approximation
- Set cover enumeration is exponential in the number of individuals



Issues with Parallel 2-Allele

- To achieve enumeration: Must intersect across all loci
- **Problem:** Number of feasible sibling groups explodes **exponentially** with the number of loci!
- **Solution:** Must cull subsets between rounds to avoid exceeding RAM and crippling performance


Parallel Enumeration Phase:
Enumerate Maximal Feasible Sibling Groups at Locus 1
Enumerate Maximal Feasible Sibling Groups at Locus 2
...
Enumerate Maximal Feasible Sibling Groups at Locus N
Set of Maximal Feasible Sibling Groups at Locus 1
Set of Maximal Feasible Sibling Groups at Locus 2
...
Set of Maximal Feasible Sibling Groups at Locus N
Parallel Intersection
Cull Subsets
Parallel Intersection ...
Cull Subsets
Parallel Intersection
Cull Subsets
Set of Maximal Feasilbe Sibling Groups at All Loci
Parallel Intersection Phase:
Generate minimal set cover

Catch 22!

- **A simple method of subset culling:**
  - To see if a sibling group is a subset of any other sibling group, *must check against all other sibling groups*

- *Notes:*
  - O(n^2)
  - Necessary for memory and performance, but very slow!
  - Can account for as high as 80% of processing time
  - Performance bottleneck, infeasible for large data sets
  - Data dependencies make parallelization a chore!

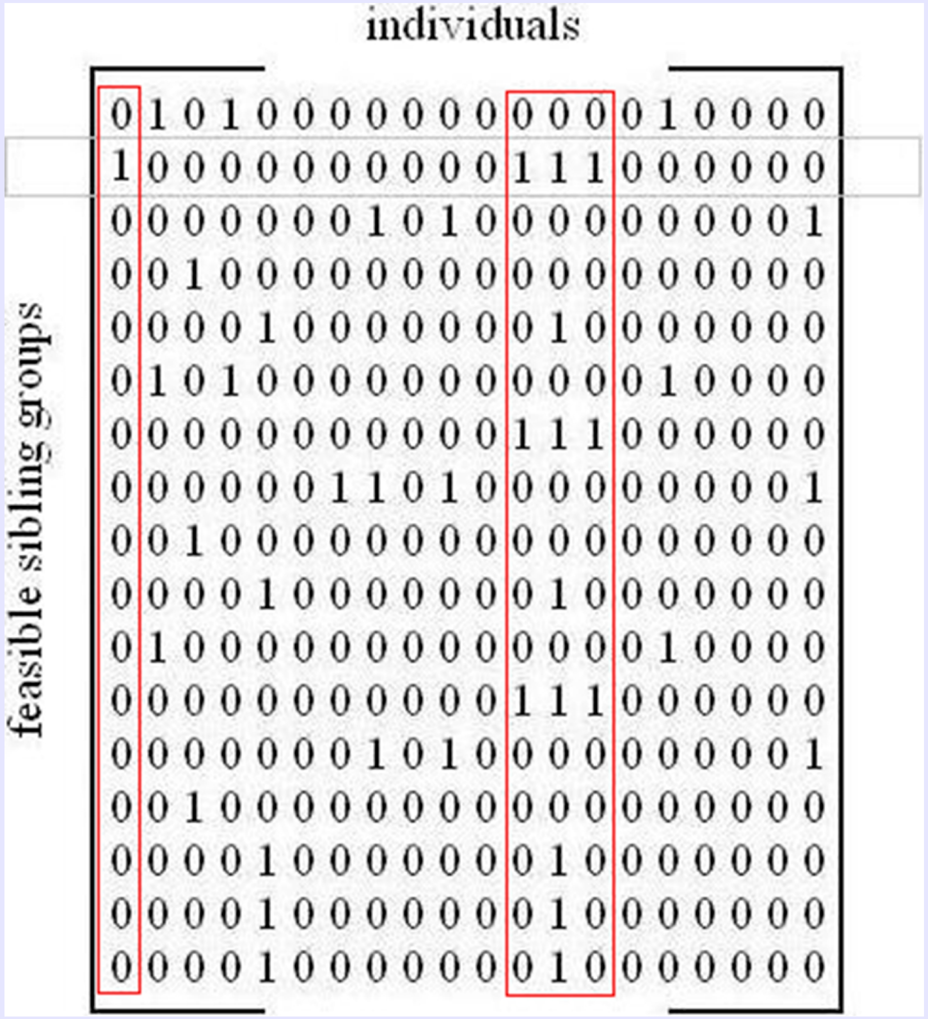
- **How do we improve on this?**

Observations

- **Matrix representation:**
  - The collection of feasible sibling groups can be represented as a bit matrix.
- **Groups are small:**
  - Each feasible sibling group contains, on average, approximately 2% of the population. In other words, the number of feasible groups, *s*, greatly exceeds the average number of individuals, *m*, which belong to any sibling group (i.e.; *s* >> *m*).
- **Most groups are supersets**:
  - In practice, 55% to 85% of feasible siblings groups are not subsets of any other sibling group, and, therefore, will not be culled.

Matrix representation

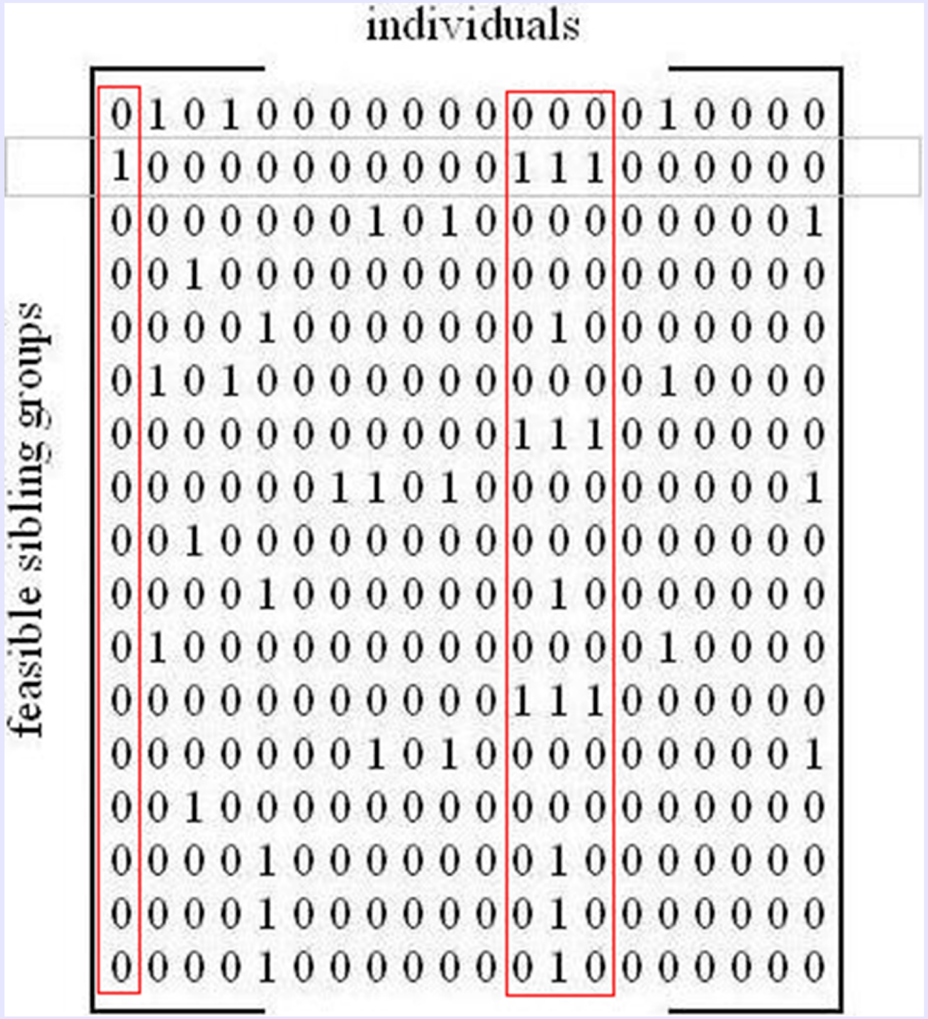
- Rows are feasible sibling groups
- A 1 in the *i*-th column means *Individual$_i$* is a member of that sibling group



$$
\begin{array}{c|c}
 & \text{individuals} \\
\hline
\text{feasible sibling groups} &
\left[\begin{array}{cccccccccccccccccccc}
0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&1\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&0&0&1\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&1\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0
\end{array}\right]
\end{array}
$$

What about columns?

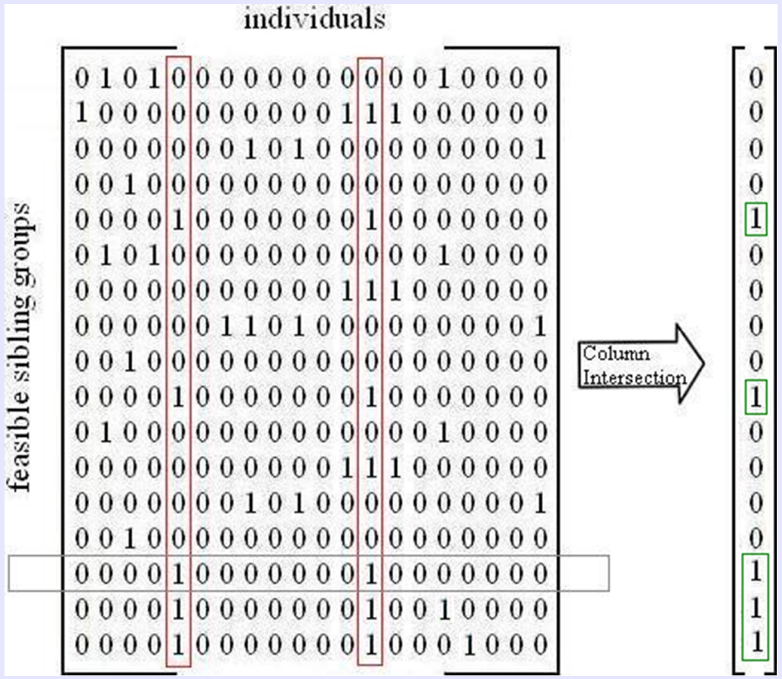
- They have meaning too!
- *Column*$_i$ defines which feasible sibling groups contain *Individual*$_i$ as a member
- What happens if we intersect the positive columns of a feasible sibling group?


individuals
feasible sibling groups

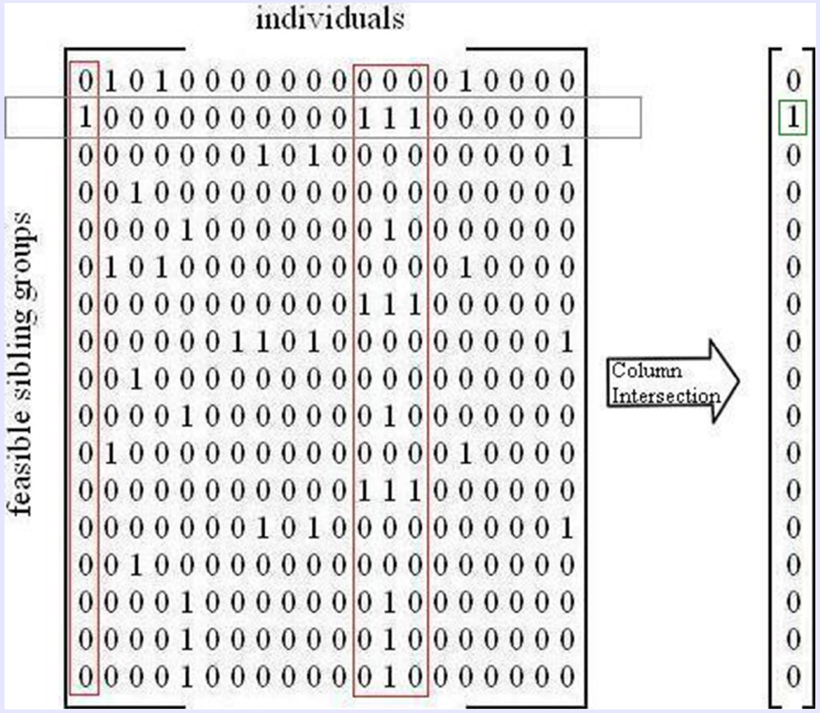
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Column-based subset culling

Case 1: A subset



Case 2: A superset



Conclusion:

The output vector can be used to classify a sibling group as a subset.

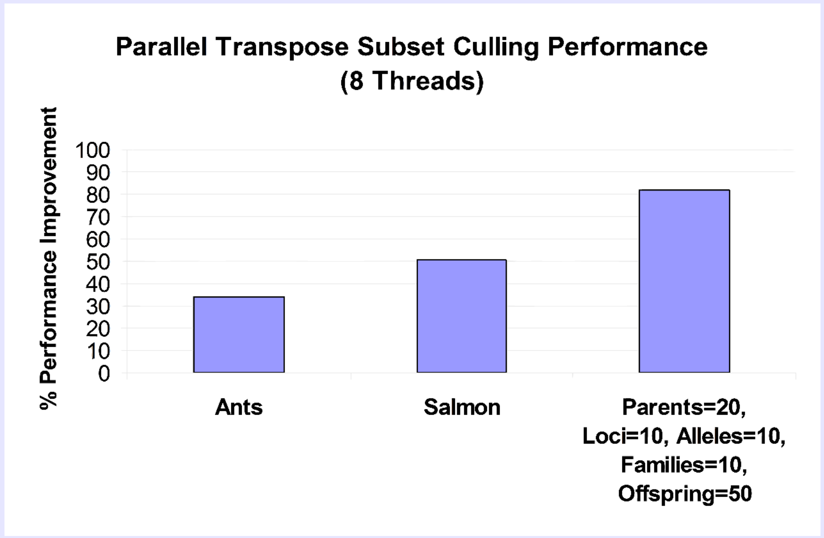
How many bits would be examined in Case 2 with the original culling algorithm?

Pseudocode

```
/* M is a bit matrix of feasible sibling groups. */
ParallelTransposeSubsetCulling(bit matrix $M$)
    /* Take the transpose of M in order to place columns
        into contiguous memory. This allows bitwise operators
         to be used on columns. */
    $T \Leftarrow$ parallelComputeTransposeOf($M$)
    parallel for each row $G_x$ in $M$ do
        /* Note: each row of M is a sibling group. */
        $y \Leftarrow G_x$.getFirstTrueBitPos()
        $Result \Leftarrow T$.getRow(y)
        while $z \Leftarrow G_x$.getNextTrueBitPos() do
            $Result \Leftarrow Result$.bitwiseIntersect(T.getRow(z))
        end while
    if $Result$.hasMultipleBitsSetToTrue() then
        markAsSubset($Result$)
    end if
end for
```

Advantages & Results

- **Easy to parallelize:** subset classification of a sibling group is independent of the classification of other groups
- **Exploits sparse nature of sibling group matrix:** Intersects only relevant columns
    - No longer need to examine every bit in the worst case (i.e. - the majority case!)

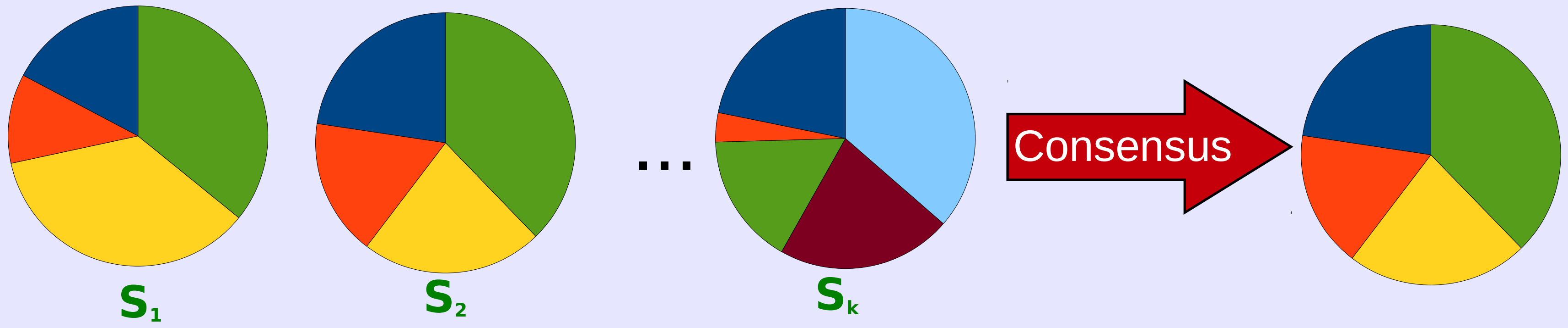


Outline

- Motivation and Background
- Improvements to Parallel 2-Allele
- **Parallel 2-Allele Consensus**
- Future Work
- Q & A

Consensus Motivation

- **Problem:** Real world genotypic data has errors and mutations!
- Solution: Compute multiple solutions and take a consensus



Consensus Algorithm

- For each *locus*$_i$, compute a sibling reconstruction with *locus*$_i$ removed.
- Solutions are then merged via strict consensus followed by a distance-based heuristic.
- What happens when 2-Allele is used?

Redundant Loci Intersections

- **Locus drop-out solution for *locus*$_i$:**
  - $\text{locus}_1 \cap \text{locus}_2 \cap \ldots \text{locus}_{i-1} \cap \text{locus}_{i+1} \cap \ldots \text{locus}_n$
- **Locus drop-out solution for *locus*$_{i+1}$:**
  - $\text{locus}_1 \cap \text{locus}_2 \cap \ldots \text{locus}_{i-1} \cap \text{locus}_i \cap \ldots \text{locus}_n$
- **Same colors are redundant intersections!**

Order Matters!

Comparison:

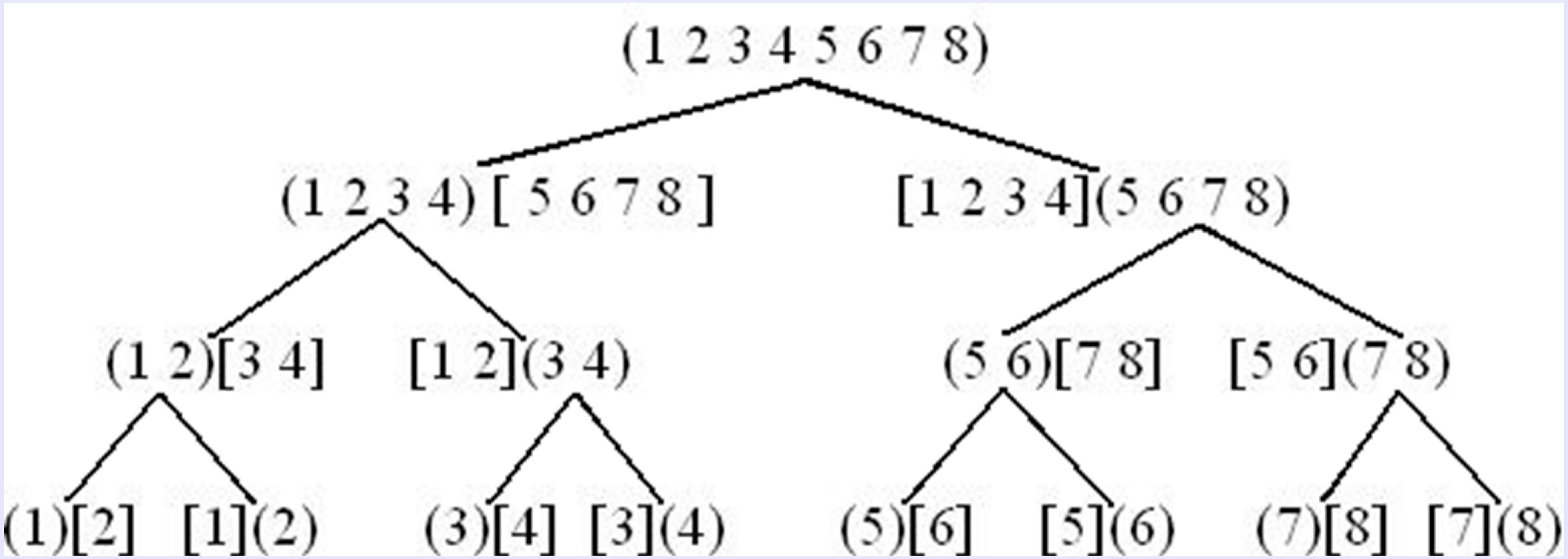
1. $((\text{locus}_1 \cap \text{locus}_2) \cap \text{locus}_3) \cap \text{locus}_4$
2. $(\text{locus}_1 \cap \text{locus}_2) \cap (\text{locus}_3 \cap \text{locus}_4)$

- **Data has shown: Method 1 performs better in practice! Why?**
  - Because of quadratic nature of an intersection round and variability in culling

    → Do not want to intersect two large collections of sibling groups with each other, you'll have to cull more as well!
  - It follows then that we want to sort loci and intersect from highest to lowest ordered

- **Conclusion**: Want algorithm that sorts loci by order, performs incremental intersections, and reduces number of redundant intersections

Recursive Approach



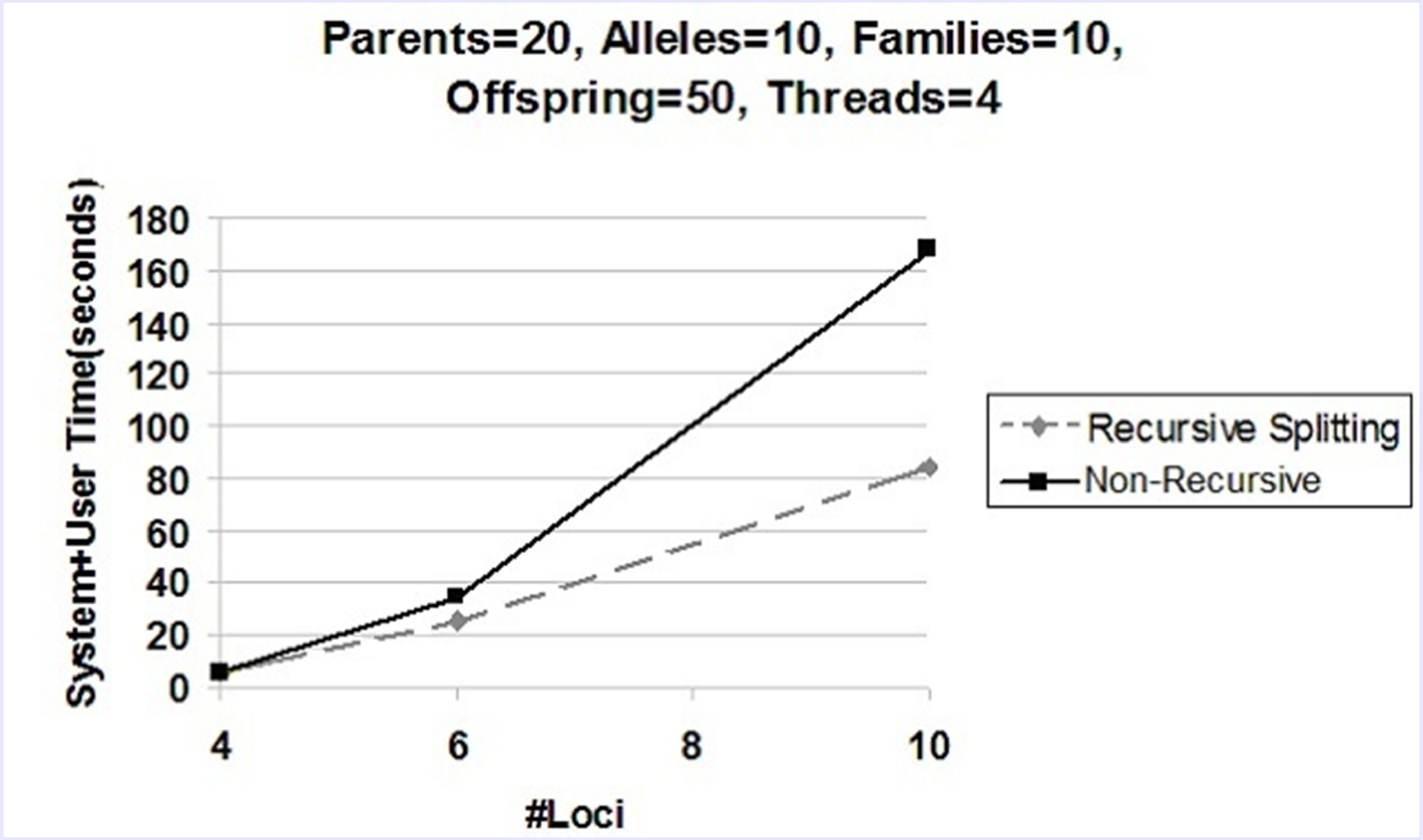
- Key is to cache all intermediate intersections!
- Cached intersections are in square brackets, includes those from parent

- To avoid intersecting large sets, as in Method 2, if at any time, the size disparity between left and right subtrees exceeds a threshold, a “quadratic method” is used on the left subtree which reuses any cached parent intersections.

Pseudocode

```
/* L is a sorted set of loci. C is a cache of loci intersections */
CompLociDropoutSolnsRecur(L, C)
    /* Base case: we only have a single locus to process. */
    if L.size() == 1 then
        outputSolution(L, C)
        return
    end if
    /* Split set of loci in half. */
    L.split(l_half, r_half )
    /* Recurse down right child tree. */
    l_cache ⇐ l_half.parallelIntersectAll(C, intermed_cache)
    CompLociDropoutSolnsRecur(r_half, l_cache)
    /* Conditionally recurse down left child tree, else perform quadratic
       procedure. */
    if shouldRecurseDownLeftChildTree(l_half, r_half, thresh) then
        r_cache ⇐ r_half.parallelIntersectAll(C, NULL)
        CompLociDropoutSolnsRecur(l_half, r_cache)
    else
        CompLociDropoutSolnsQuadrat(l_half, intermed_cache)
    end if
```

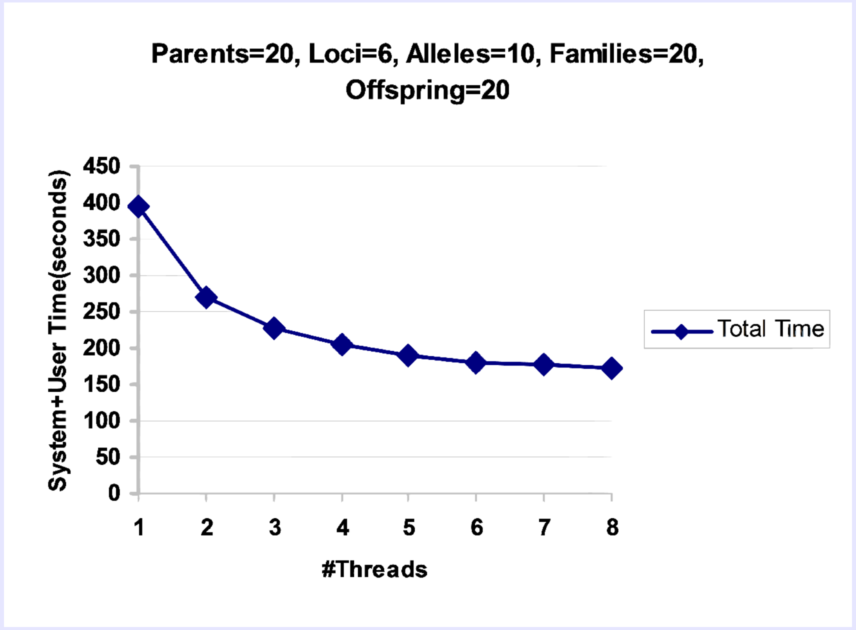
Results



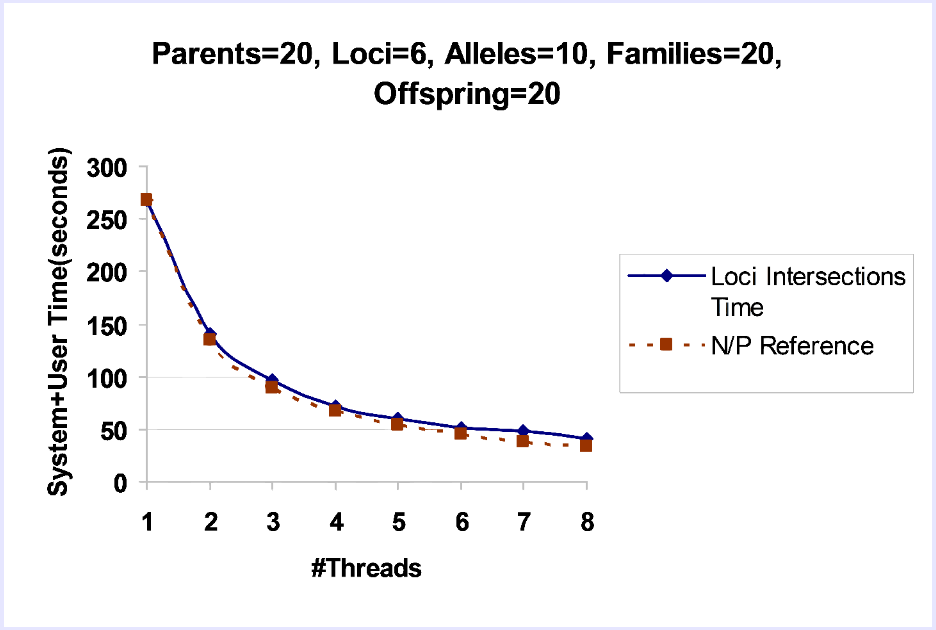
Parents=20, Alleles=10, Families=10, Offspring=50, Threads=4
System+User Time(seconds)
180
160
140
120
100
80
60
40
20
0
4
6
8
10
#Loci
Recursive Splitting
Non-Recursive

Results

Total Parallel Consensus Reconstruction Time



Parallel Loci Intersection Time



Outline

- Motivation and Background
- Improvements to Parallel 2-Allele
- Parallel 2-Allele Consensus
- **Future Work**
- **Q & A**

Future Work

- **More cores!** MPI and GPU implementations
- **Sparse Intersections**: Need method to exploit sparseness of matrix columns and not just rows

Questions?