CS392F: Automatic Software Design

P7: Gamma Joins

Jianyu Huang Xiaohui Chen

[jianyu@cs.utexas.edu](mailto:jianyu@cs.utexas.edu) [xhchen0328@utexas.edu](mailto:xhchen0328@utexas.edu)

# Requirement for the write-up

Explain and demonstrate that you have implemented the final parallel hash-join architecture in the lecture notes. You should have substantial tests for each component that you write. And you should have a substantial write-up of what you have done, how you have proceeded, and your organization.

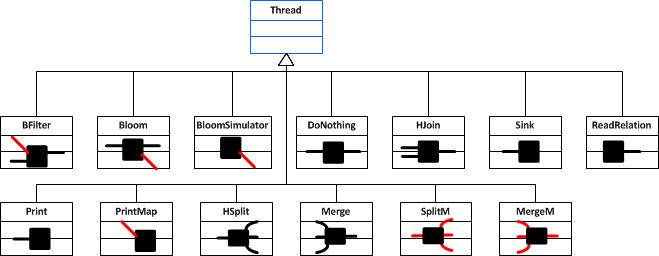
# Running instructions

Please execute the bash-script in Cygwin.

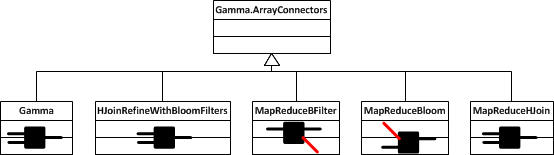
|  |
| --- |
| 1. Make the the script executable  *chmod u+x run.script.sh*  2. Run the script  *bash run.script.sh* |

# Code Organization

Following the step-wise approach instructions of programming assignment, our code organization looks like the followings:



On top of the above boxes, we implement pipe-and-filter graphs that implement the functionality of the primitive boxes. These boxes implement various non-primitive implementations of the HJoin box/primitive. The terminal boxes are subclasses of Gamma.ArrayConnectors, which provides useful ways of creating arrays of pipes in one call.



# Implementation Details

## ReadRelation

We start by implement a ReadRelation component that reads a text file and produces a stream of Strings that represent records. We decide to use regular expression and split() function to parse lines of input database files.

## Print

We implement a Print component that prints a stream of tuples to standard out.

## HJOIN, BFILTER, etc

Hash join takes 2 streams of tuples (𝐴,) as input and produces the join of these streams (A⋈B), BFILTER use bloom filter to filter part inputs so that the input domain for the later join operation will become smaller.

HJOIN and BFILTER both looks like a two-inputs one-output black box as the followings:



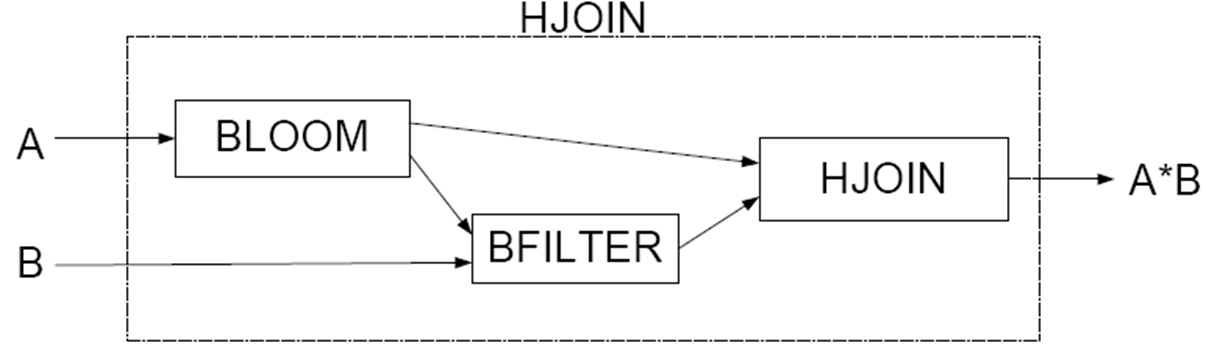


Following the instructions in the slides, we implement HJOIN and BFILTER.

For HJOIN, first we read all of stream A into a main memory hash table, then we read B stream one tuple at a time and finally hash join key of B’s tuple and join it to all A tuple’s with the same join key. We use linear algorithm in that each A,B tuples is read only once.

For BFILTER, The filtering part of Bloom filters eliminates B tuples that cannot join with A tuples. First we read bit map M, then we read each tuple of B, hash its join key: if corresponding bit in M is not set, we simply discard tuple (as it will never join with A tuples), otherwise, we output tuple.

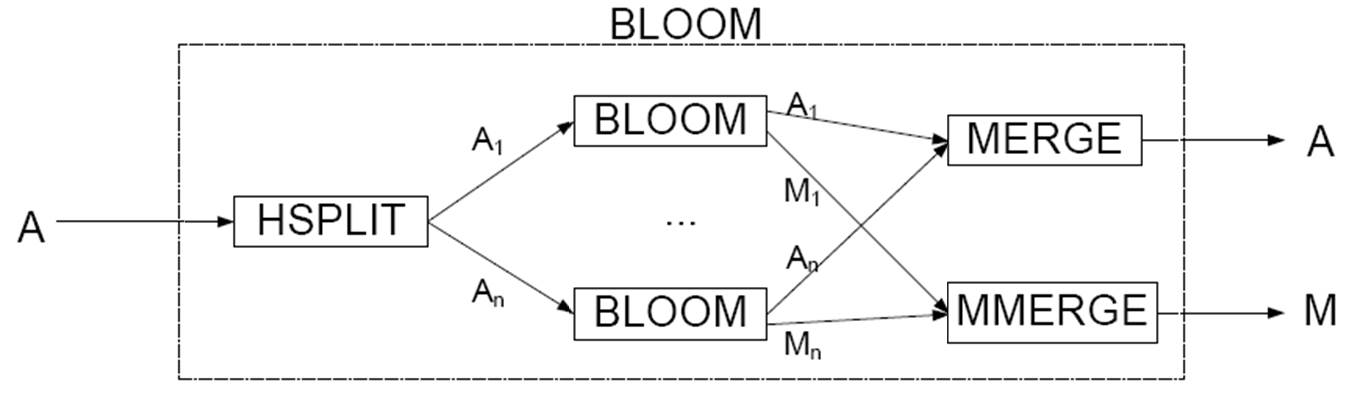
So the initial unoptimized version of gamma filter looks like the following:



# Implementation of the final parallel hash-join architecture

## Parallelization of BLOOM Box

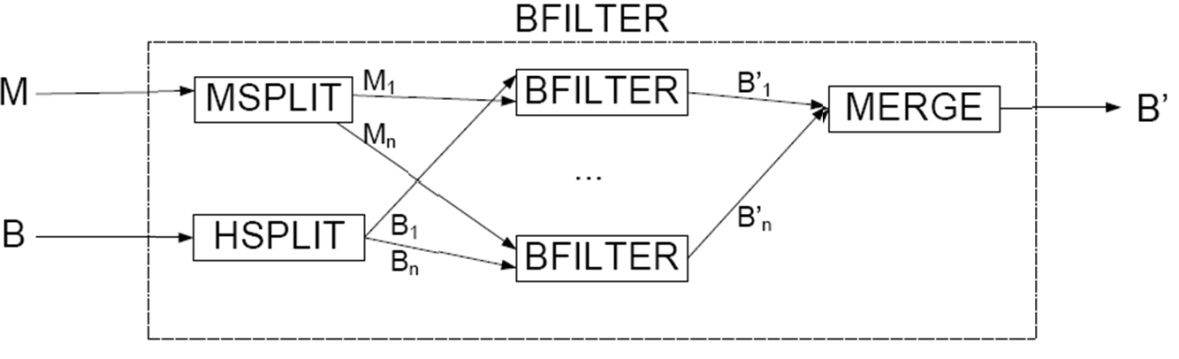
Following the instructions in the slides, we parallelize BLOOM Box as the followings:



We first HSPLIT stream A and compute Bloom filter on each substream, then reconstitute stream A by forming merge bit maps to produce single bit map M.

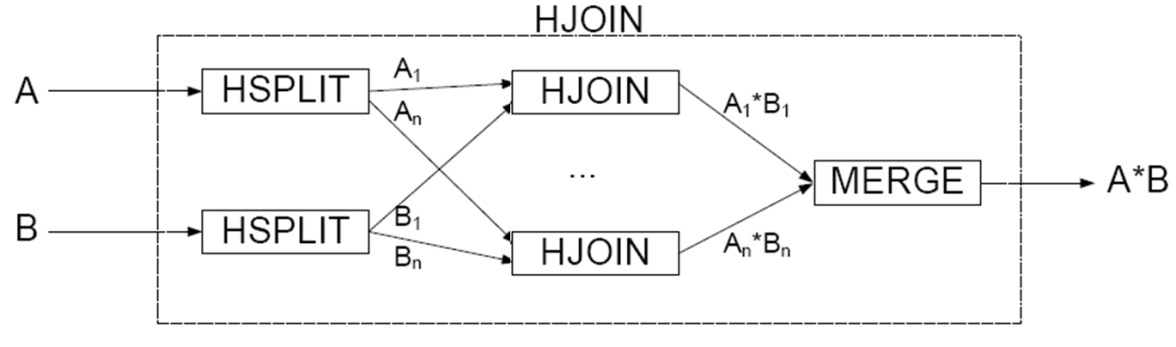
## Parallelization of BFILTER Box

Following the instructions in the slides, we parallelize BFILTER Box as the followings:



We split *M* into *M1*…*Mn*, and hash split stream *B* into *B1*…*Bn*, then we filter *Bi* substreams in parallel, finally we reconstitute stream *B’*.

## Parallelization of HJOIN Box



We split both streams using same hash function so that A and B tuples can join only if they have the same hash key, then we perform *n* joins (rather than *n2*) in parallel, finally we reconstitute the join operation and merge the results.

## Final Design

The final design of Gamma Join looks like the following:

