- **Introduction**
  - **band join**
    - » **$R.A - c1 < S.B < R.A + c2$**
  - **partitioned band join**
    - » **goal : minimize number of disk accesses**
  - **Sort-merge band join**
    - » **combining final merge phase with join phase**
- **Partitioned band join algorithm**
  - **choose partition size**
    - » **randomly sampling R**
    - » **select partition elements**
      - ✓**Kolmogorov test statistic [W.J.Conover 71]**
    - » **number of partitions**
  - **perform partitioning without sorting**
    - » **use range-vector**
    - » **overlap**
      - ✓**$Li - c1 < S.B < Hi + c2$**
  - **compute subjoins between Ri and Si**
    - » **binary search on sorted inner partition**

- **Uniprocessor environment**
  - **equal relation sizes : figure 1.**
  - **relation sizes differ : figure 2.**
  - **GP : grace partitioned band join algorithm**
  - **HP : hybrid partitioned band join algorithm**

- **Multiprocessor environment**
  - **parallel hybrid partitioned band join**
    - » **each processor randomly samples**
    - » **coordinator determines partitioning elements**
    - » **each processr re-distributes local fragment**
  - **problem**
    - » **how to correctly and efficiently sample inner relation in parallel to determine partitioning elements**

- **Experiment results**
  - **scaleup : slight increase in response time**
    - » **duplicate initiating tasks**
    - » **effects of short-circuiting messages diminish**
    - » **skew in size of subjoins**
      - ✓ **sampling time vs. execution time**
  - **speedup : not perfectly linear**
    - » **overhead of scheduling operators**
    - » **same factors as cases in scaleup**

- **Conclusions**
  - **hash-based equijoin vs. partitioned band join**
    - » **hash bucket vs. sort inner partition**
    - » **table lookup vs. binary search**
    - » **sampling overhead**
  - **suitable cases**
    - » **a fraction of relation fits in memory**
    - » **relation sizes are different**

- Introduction
  - multiple-query scheduling
    - optimize a set of queries together
    - share some operations within queries
    - examples : shared build or shared probe

- Sharing operators
  - select
    - apply each predicate in turn for each tuple
  - join
    - hybrid hash-join algorithm
      - ✓sharing build phase
    - multiprocessor join algorithm
      - ✓joins involving same relation on different join attributes
  - sort, aggregates, group-by, duplicate elimination

- **Batch scheduling**
  - **aim : find a global schedule for all queries without violating partial order constraints**
  - **workload : single hash join queries**
  - **cost metric : total number of I/Os**

- **Algorithms**
  - **Non-sharing algorithm**
  - **Exhaustive algorithm**
    - » **select a global ordering of queries**
    - » **choose building relations**
    - » **determine order of flush of relations in memory**
  - **Heuristic algorithms**
    - » **select next building relation**
      - ✓**ranking functions : ProbSize, ProbSubBuild, ProbDivBuild**
    - » **select next relation to flush**
      - ✓**Largest, LowRank, HighRank**

- **Simulation**
  - **shared-nothing model**
  - **comparison**
    - » **ratio of batch size to number of relations : figure 1.**
      - ✓**memory requirement**
      - ✓**replication tuples**
    - » **sensitivity to system memory : figure 2.**
      - ✓**naive algorithm gains due to parallelism**
    - » **non-uniform relation usage : figure 3.**
      - ✓**skewed distribution has higher potential for sharing**

- **Future work**