**Garbage Collection to Manage Future Memory Better**

**Research Goal**
- Future memory will need efficient GC
- Most of the existing GC algorithms need to be changed or improved
- My goal is to build high performing GC for future memory management
- Reference counting is important but not well understood for modern architectures

**Future Memory Management**
- Today processor is in the center of computing, and information is moved to be used in computation and then stored
- Communication between processor and memory will be different in future computer design where major concerns are power and performance
- New approach would be to marry processing to memory to cut down transportation of data and reduce energy use
- In future, memory bandwidth contention will be a major bottleneck due to increasing hardware parallelism

**Reference Counting (RC)**
- For each object maintains a count of number of references to it
- When the count becomes zero, the object is garbage and can be collected
- Helps ensure program integrity
- Automatically reclaim memory that is no longer in use
- Relieves the burden of explicitly freeing allocated memory which may be tricky

**Approaching the Goal**
- Study of two major design points of reference counting
  - Storing the count
  - Collecting cyclic objects
- Analysis of reference counting intrinsic
- Design and evaluation of different mechanisms to store the count

**Reference Count Distribution**
- Reference counting needs space to store the count
- Distribution of the count for different benchmarks is useful
- Most of the objects reference counts are in between 1 to 6

**How to Store the Count**
- Using an extra word in the object header
  - Widely used but may not be space efficient
  - Using some limited unused bits in the object header
    - Some objects count may overflow and they are termed as stuck objects
    - Need separate strategies to handle stuck objects

**Stuck Object Statistics**
- Minimum 2 bits and maximum 5 bits can be used if we don’t use the extra word
- Percentage of stuck objects will change if we vary the number of bits used

**Collecting Cyclic Objects**
- Reference counting cannot collect cyclic objects
- Needs separate mechanism like backup tracing
- Benchmarks like javac, mpegaudio, eclipse, hadoop, lusearch and xalan have significant cyclic garbage

**Strategies to Handle Stuck Objects**
- **RCTable** – Use an auxiliary data structure (i.e. hash table) to store the overflowed reference counts of the stuck objects
- **RCIgnore** – Ignore the stuck objects and depends on the backup trace to collect them
- **RCRestore** – Use backup trace to restore the reference count of the stuck objects so that they can be collected by reference counting

**Evaluation of the Strategies**
- RCTable performs worse than standard reference counting (RefCount) with respect to both total time and total GC time
- RCIgnore and RCRestore both performs better than standard reference counting (RefCount) with respect to both total time and total GC time

**Conclusion**
- As RCRestore performs best, limited unused bits can be used to store the count and backup tracing can be used to restore the count of the stuck objects
- Understanding of reference counting for modern architecture is quite clear now
- It also happens to be first quantitative analysis of reference counting design points