Memory Analysis and CPU-time Profiling in RMG-Java

Kehang Han
Jan. 22, 2014
Outline

Memory Management in Java

Demo of Memory Analysis

CPU-time Profiling

Possible Approaches
Memory Management in Java

• Programming languages like C/C++
  o Manually allocate/de-allocate memory

• Java
  o Automatically de-allocate
  o Garbage collector
Basic concepts for Garbage Collection

- **Heap dump**
- **Shallow heap**
  - Memory consumed by one object itself
- **G.C. root**
  - Any variables your program can access directly
    - Local variables
    - Class static variables
Basic concepts for Garbage Collection

- Live objects: Can be reached from G.C. Root
- Retained set & heap
- Dominator
Mark and Sweep Garbage Collection

Marking

Before Marking

After Marking

- Alive object
- Unreferenced Objects
- Memory space
Mark and Sweep Garbage Collection

Normal Deletion

After normal deletion

Memory Allocator holds a list of references to free spaces, and searches for free space whenever an allocation is required
Mark and Sweep Garbage Collection

Deletion with Compacting

After normal Deletion with compacting

Memory Allocator holds the reference to the beginning of free space, and allocated memory sequentially then on.
Outline

- Memory Management in Java
- Demo of Memory Analysis
- CPU-time Profiling
- Possible Approaches
RAM limitation

Plot of Memory Percentage along Running Time for all versions

- v101
- v201
- v301
RAM limitation
Demo of Memory Analysis

• How to get a heap dump
  o Console: jmap -dump:format=b,file=<filename.hprof> <pid>
  o .sh file: -XX:+HeapDumpOnOutOfMemoryError

• How to use Eclipse Memory Analyzer
  o Histogram
  o Outgoing & incoming
  o Dominator tree & immediate dominator
  o Retained set
Object Graph in RMG-Java

D is a core species? ✗
D is a new edge species? ✓
Demo of Memory Analysis

ChemGraph is the class of objects that occupy most RAM!
What ChemGraph Dominates?

- Graph: 89%
- String: 5%
- ThermoData: 4%
- TransportData: 2%
Outline

- Memory Management in Java
- Demo of Memory Analysis
- CPU-time Profiling
- Possible Approaches
CPU-time Profiling

```java
// ENLARGE THE MODEL!!! (this is where the good stuff happens)
pt = System.currentTimeMillis();
enlargeReactionModel();
double totalEnlarger = (System.currentTimeMillis() - pt) / 1000 / 60;

//**********333***********% initialize PDepNetwork
pt = System.currentTimeMillis();
// 10/24/07 gmagoon: changed to use reactionSystemList
if ((reactionModelEnlarger instanceof RateBasedPDepRME)) { // 1/2/09 gmagoon and rwest: only call PDepNetwork for P-dep cases
    for (Iterator iter = reactionSystemList.iterator(); iter.hasNext();)
        { System ReactionSystem rs = (ReactionSystem) iter.next();
            rs.initializePDepNetwork();
        } reactionSystem.initializePDepNetwork();
    dtDep = (System.currentTimeMillis() - pt) / 1000 / 60;
```
CPU-time Profiling

- Enlarging model: 33%
- PDepNetwork: 33%
- Solving ODE: 23%
- Writing file: 11%
Outline

- Memory Management in Java
- Demo of Memory Analysis
- CPU-time Profiling
- Possible Approaches
Approach 1: Memory Usage Reduction

At later stage of reaction generation:

- ChemGraph takes up most memory,
- >95% ChemGraphs are for edge species.

Most ChemGraphs occupy memory but contribute little

Proposed approach:

- Replace edge’s ChemGraphs with much cheaper identifiers
- One identifier < 100 bytes, while one ChemGraph ~ 10^4 bytes,
- Can retrieve ChemGraphs back when needed,
- Can compare with other edge species using identifiers.
One iteration from view of **MEMORY**

```
Edge

A ← B
Core

C → F

G

D

Dynamic simulation

Edge

A ← B
Core

C → F

G

D

Species selection

Edge

A ← B
Core

C → F

G

D

Rxn generation

Edge

A ← B
Core

C → F

G

D
```
Upon Reaction Generation

Reaction model
  ↓
  Core
  ↓
coreRxn  coreSpe
  ↓
Edge
  ↓
edgeRxn  edgeSpe

Rxn1
  ↓
structure  kinetics
  ↓
reactants  products

(Species)
  ↓
crSpe A  crSpe B

(chemGraph)
  ↓
cg1  cg2  cg3  cg4

Rxn2
  ↓
structure  kinetics
  ↓
reactants  products

edSpe D
  ↓
cg5  cg6
In original design, **Dynamic simulation** is the next step;

Now new steps added **BEFORE** that:

**Memory Usage Reduction Method**
ChemGraph $\rightarrow$ SMILES

Reaction model

Core

- coreRxn
- coreSpe

Edge

- edgeRxn
- edgeSpe

Rxn1

- structure
- kinetics

- reactants
- products

(Species)

- crSpe A
- crSpe B

Rxn2

- structure
- kinetics

- reactants
- products

(Species)

- edSpe D

(EdgeSpecies)

- edSpe D

(SMILES)

- s5
- s6
If edge species D is a **new one**

**Garbage collected!**
Now comes Dynamic simulation & Selection

**Edge species D** will be finally entering core
Upon Species D being selected
Approach2: Pruning Edge Species

Pruning will be done based on fluxes.

- Upper limit of edge species
- Below a certain flux
Approach3: Job Partition

Heavily limited by the 10K edge species
How to Partition Job

• Each processor **keeps a copy** of core model in its own memory;
• Edge species **almost evenly split** into N pieces for N processors;
  o Using M.W. makes partition easy and fast
  o Processor1 collects those species with M.W. ≤ 30
  o Processor2 collects those with 30 < M.W. ≤ 60
  o ......
• Edge reactions **go where corresponding** edge species go;
  o e.g. CH3 + C2H6 → CH4 (M.W.=16) + C2H5 (M.W.=29)
  should go to Processor1
  o e.g. CH3 + C2H5OH → CH4 (M.W.=16) + C2H5O (M.W.=45)
  Processor1 stores CH3 + C2H5OH → CH4 (M.W.=16) + “other edgeSpecies”
  Processor2 stores CH3 + C2H5OH → C2H5O (M.W.=45) + “other edgeSpecies”
How Job Runs Differently

Step 1: ODE solving. (Not affected)
- Edge species *don’t serve as reactants*
- Core species and edge species are *decoupled* in ODE system
- ODE solver in each processor stops *at different conversion*

Step 2: select new core species. (Need communication)
- Processor with *smallest conversion*

Step 3: update core and edge model.
- Move the new core species from edge to core
- Move related edge reactions to core except those having “other edgeSpecies”
- Make reactions between new core species and old core species
  - Not all products are core species → checking where to go
  - All products are core species → checking reverse reactions
How Job Runs Differently

[Graph showing conversion vs fluxes with different curves and a threshold curve]