Cost of Implementing Final Fields

Memory Barriers

- To ensure final field immutability, requires membar between construction and read of field on reader & writer sides
- membar = 30 cycles on 21164 Alpha
- On 400 mHz machine = 75 nanosecs
- Test Setup
  - Sun Ultra 60 (OK, is a cheat)
  - Finalized SPEC benchmarks

Projected Slowdown on (finalized) SPECjvm98

Benchmark	Seconds	Seconds/MB
Compress	33	120 (x3.6)
Db	42	52 (x1.2)
Jack	17	28 (x1.6)
Javac	32	35 (x1.1)
Jess	15	21 (x1.4)
Mpeg	37	67 (x1.8)
Mtrt	25	30 (x1.2)

getfields/getstatics/aaloads of finals

- compress:1,154,641,140
- db:127,964,512
- jack:144,184,226
- javac:33,309,513
- jess:72,481,686
- mpeg:397,994,634
- mtrt:66,610,552

getfields/getstatics/aaloads

- Optimized so that there is only one mb for a given object in a method
  - Maximum we can hope for from data flow analysis
  - Avg of 60% speed up, but still ugly
- compress:225,926,010 (-81%)
- db:64,563,485 (-50%)
- jack:13,024,896 (-91%)
- javac:18,500,829 (-45%)
- jess:30,442,641(-58%)
- mpeg:14,440,020 (-97%)
- mtrt:65,999,754 (-1%)

Object Aging

- Why look twice at objects?
  - Can have a nursery for new objects where you do MBs
  - Can have an "older area_ where you do not do MBs
- Can accomplish in a couple of ways

Methods

- Execute Global Memory Barrier (GMB)
  - Execute a GMB whenever a getfield of a final field of a new object is performed
  - Execute a GMB at each context switch
  - Execute a GMB whenever *n* getfields of final fields of new objects are performed
    - For other *n-1*, execute local membars

Method 1

- If a GMB is executed every time there is a getfield of a final field of a new object
  - Also "ages" any other objects created recently
  - Since they are GMBs, cannot compare directly to MB costs
  - But we get an order of magnitude or two

Results

- compress:2,299 (x500000)
- db:1,473,201(x90)
- jack: 2,843,324 (x50)
- javac:1,375,102 (x30)
- jess:1,490,406 (x50)
- mpeg:2,542 (x160000)
- mtrt:196,403 (x330)

Method 2

- Further refinement:
  - Getfield of a final field with a reference to it stored in the heap
  - If it is not in the heap, then it is local, and we do not need to perform the MBs
  - Done in addition to dataflow
- Might be difficult to detect references stored in heap
  - But let_s look at results anyway

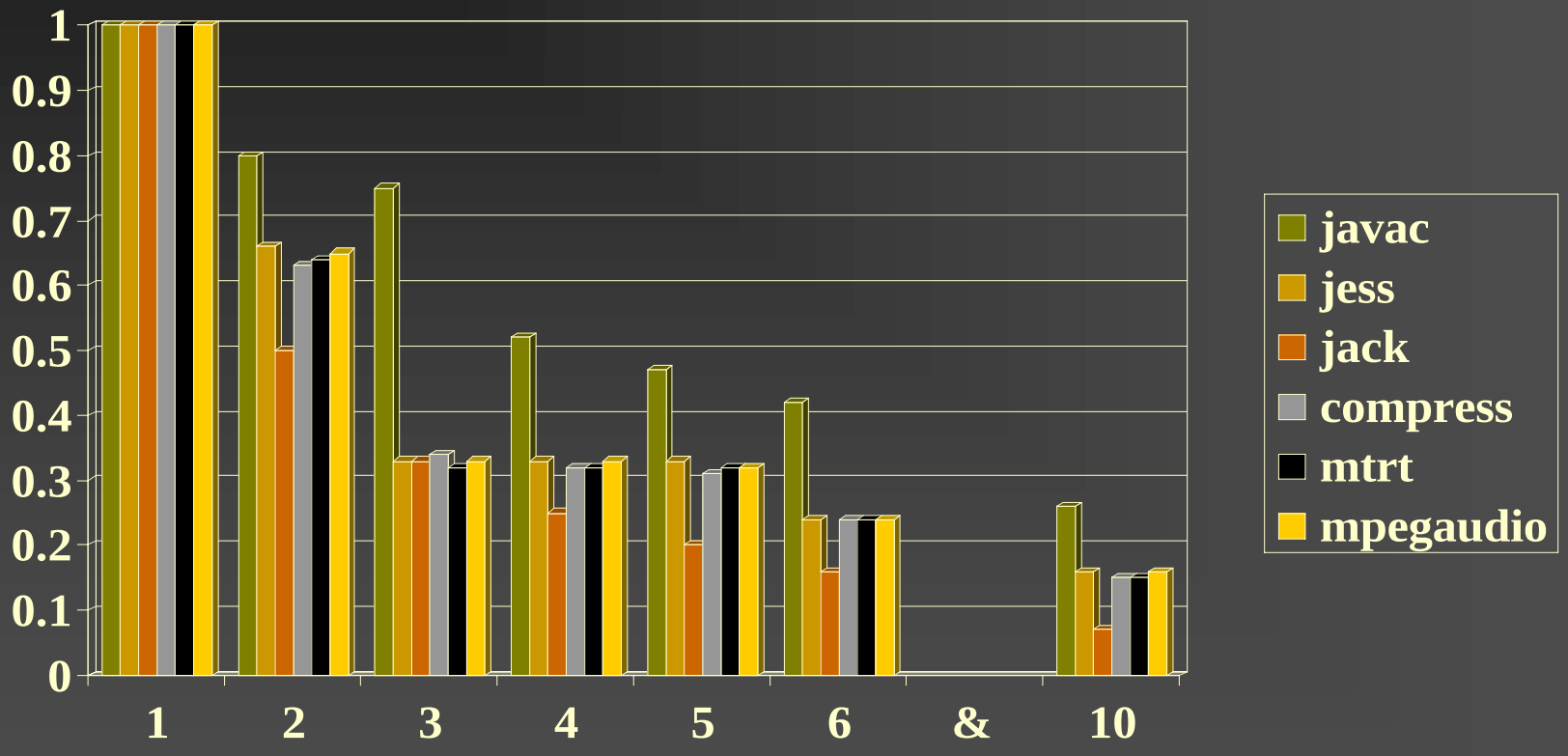
Results

- compress:125 (x920000)
- db:64 (x2000000)
- jack:3,261 (x44000)
- javac:121,942 (x270)
- jess:776 (x93000)
- mpeg:91(x4400000)
- mtrt:400 (x170000)

Method 3

- Why have a global memory barrier each time?
  - Might have significantly fewer if we had a global memory barrier every *n* accesses of a new object
  - Every other access we have a local MB
  - Would optimize *n* for GMB time vs. MB time

Performing a GMB after X MBs GMBs



Cost of Performing MBs and GMBs

- Depends on the system
- Number of MBs is roughly a multiple of number of GMBs
  - Performed after *n* membars, it is *n-1* times number of GMBs
  - Could tune performance based on comparative cost of GMB on a given system

Method 4

- What if we did it on every context swap, instead of every *n* mbs?
- Simulated by
  - counting instructions for a benchmark
  - dividing by time to get n
  - issuing a GMB every n instructions
- Results are fairly good, but a few degenerate cases

Results

- number of GMBs
- compress:125
- db:123
- jack:1,330,470
- javac:656
- jess:1,155
- mpeg:220
- mtrt:219

- number of MBs
- compress:138
- db:30,466,705
- jack:0
- javac:602,764
- jess:42
- mpeg:22
- mtrt:56

Ultimately

- The cost of implementing final field immutability in an obvious way would be excessive
- Must have a few tricks and tweaks to make finals reasonable