Reducing Compilation Overhead in J9/TR

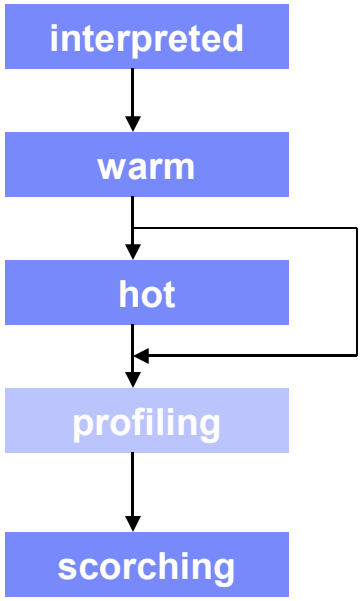
Marius Pirvu, Derek Inglis, Vijay Sundaresan

October 17, 2005

Agenda

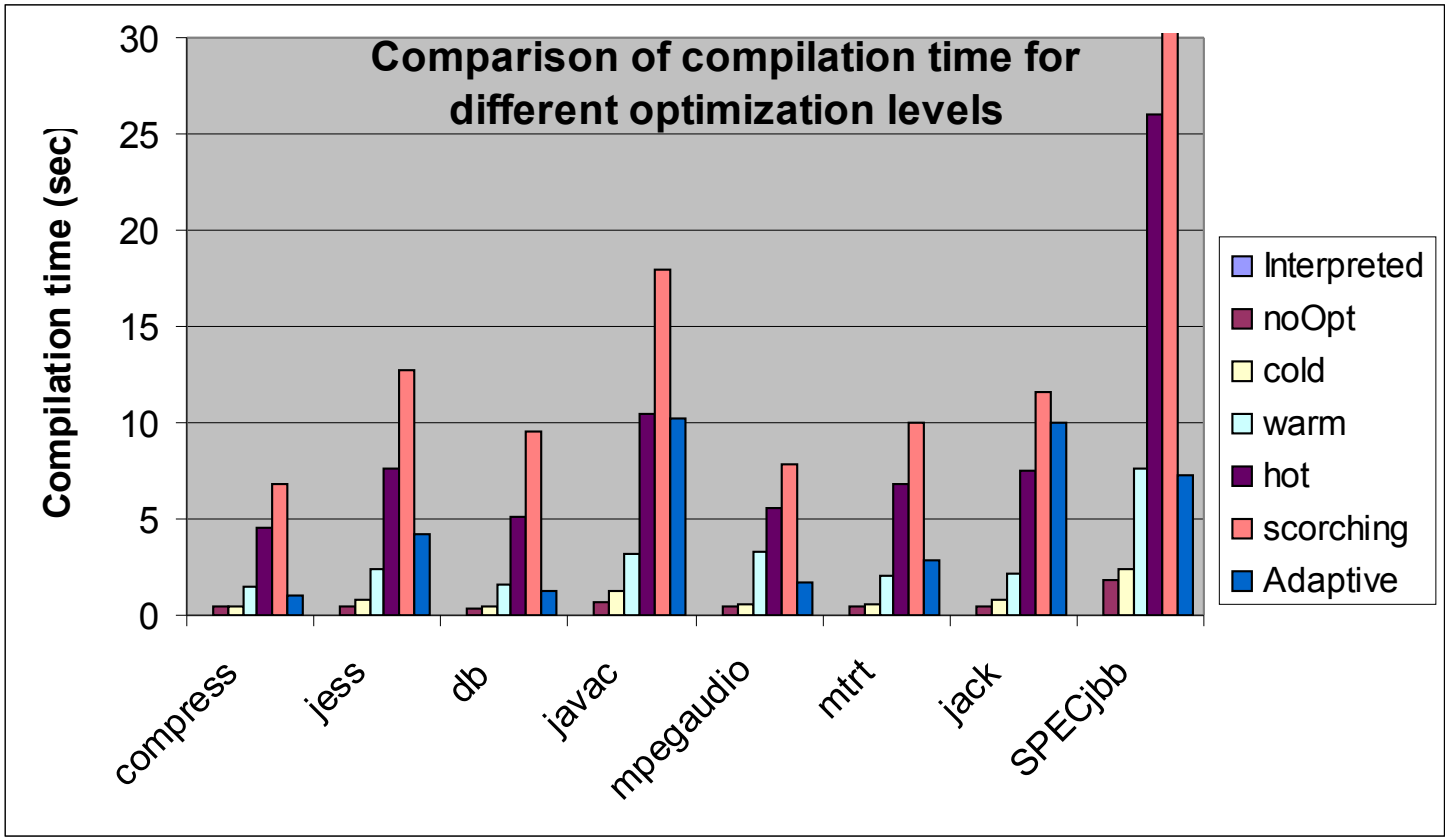
- Adaptive compilation in the TR JIT
- Class Load Phase
- Asynchronous compilation
- Limiting the negative effect of very long compilations
- Ongoing work
  - Improvements to async compilation
  - AOT

Adaptive Compilation in TR JIT

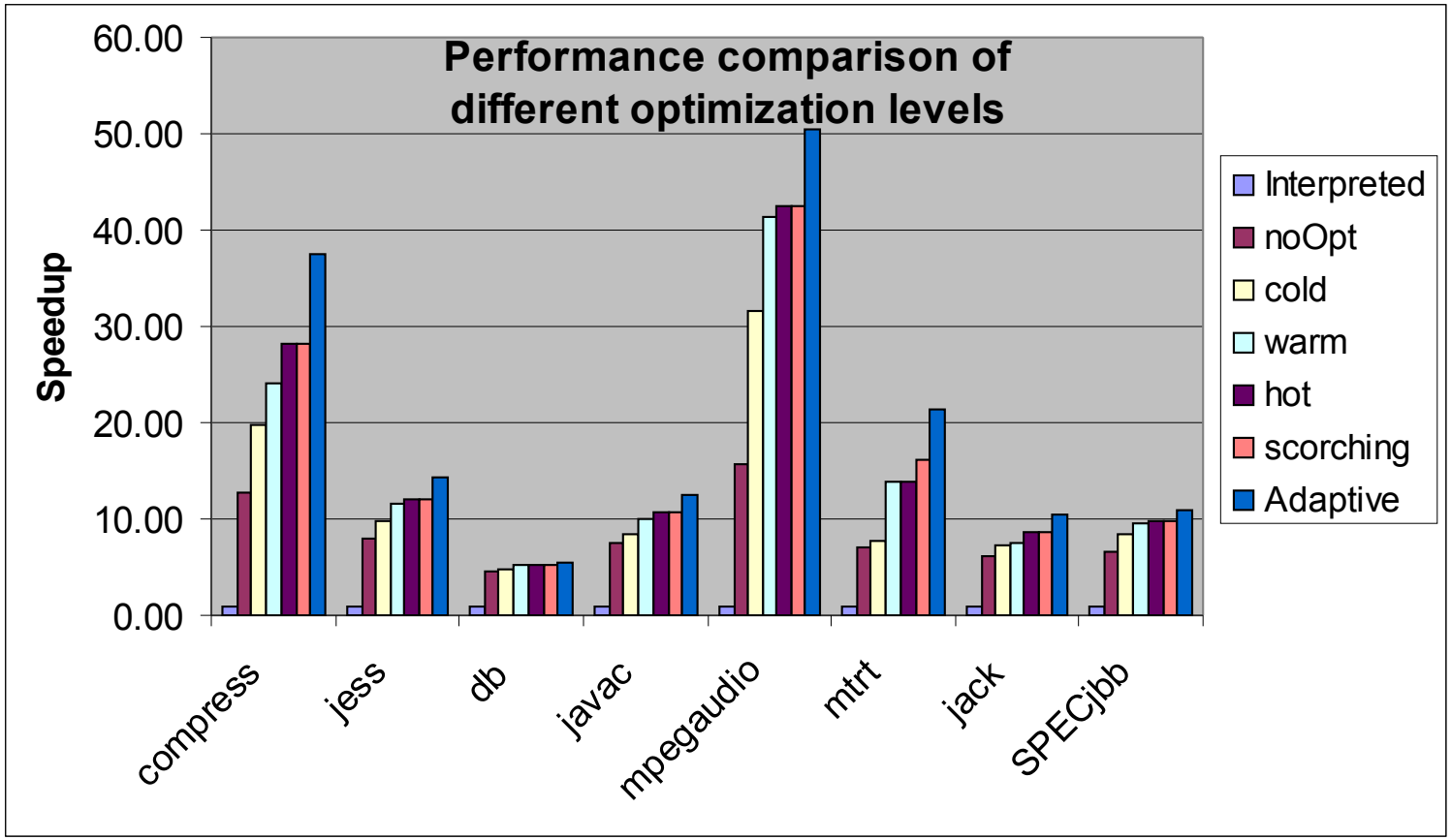


- Methods start out being interpreted
- After N invocations methods get compiled at ‘warm’ level
- Sampling thread used to identify hot methods
- Methods may get recompiled at ‘hot’ or ‘scorching’ levels
- Transition to ‘scorching’ goes through a temporary profiling step

Performance



Performance



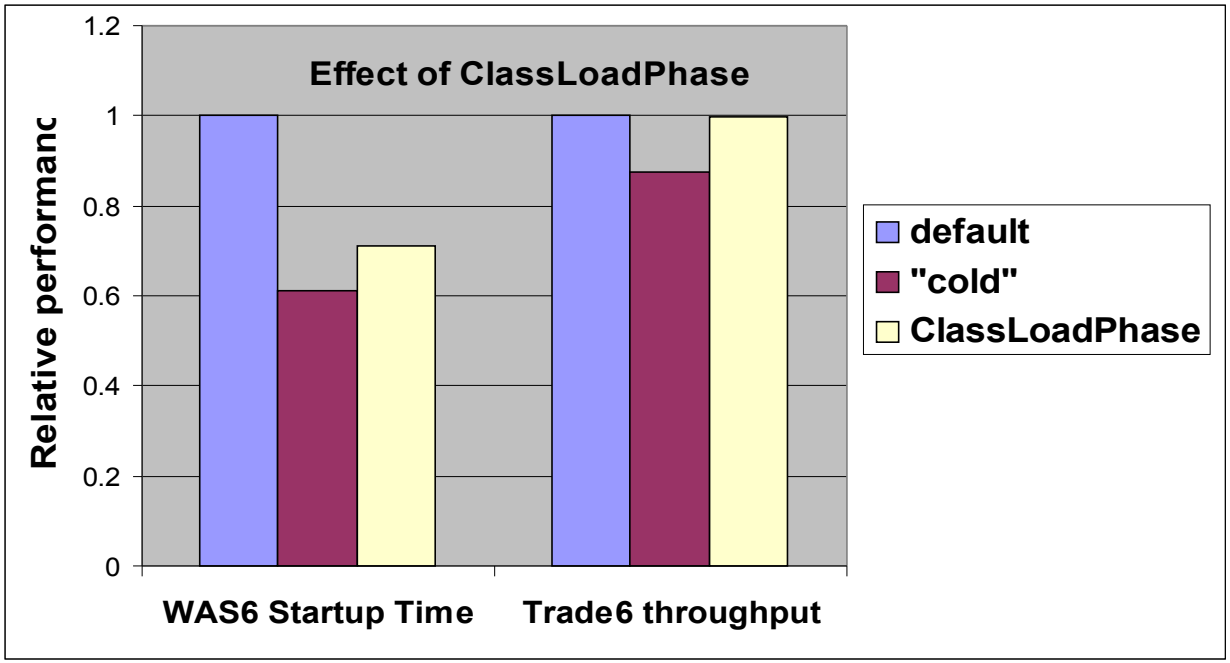
How About Applications Without Hotspots

- WebSphere AppServer startup
  - Very flat profile

	No Adaptive Compilation	With Adaptive Compilation
Methods compiled	Warm=36700	Warm=2750, hot=11
Time spent compiling	115 sec	17.3 sec
Startup-time	161 sec	21.5 sec

Class-Load-Phase

- Intuition: Methods compiled during startup phase may not be important during application run phase
- Detect phases when class loading is intense
- Reduce optimization level to “cold” during such phases



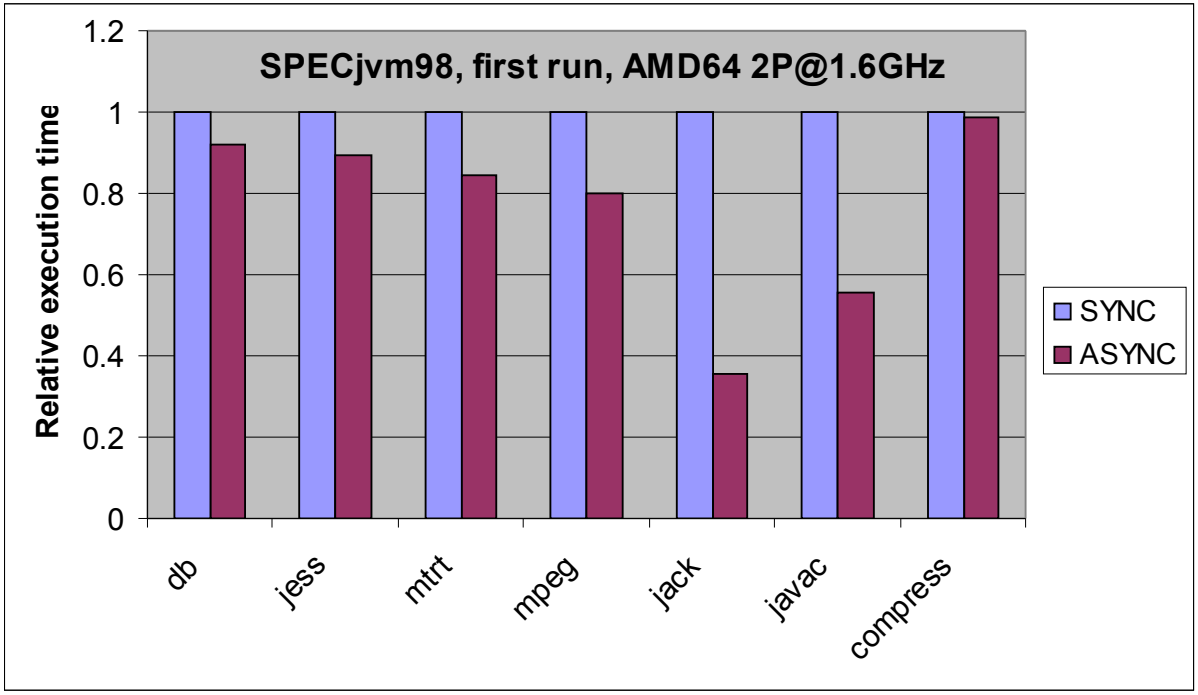
Asynchronous Compilation

- Synchronous compilation
  - Application thread places compilation request and blocks waiting for the compilation to finish
- Asynchronous compilation
  - Application thread does not wait for the compilation result
- JIT compilations performed on a separate compilation thread

Asynchronous Compilation

- Implementation
  - Synchronous compilation still needed in some cases (e.g. pre-existence)
  - Synchronous and asynchronous compilation must coexist
  - Queue of compilation requests
- Advantages
  - Takes advantage of available processors on SMP machines by increased parallelism
  - Allows performance improvements in uniprocessors by changing compilation thread priority
  - Allows reordering of compilation requests

Performance Results – Short Running Apps.



- Asynchronous compilation on SMP reduces execution time of short benchmarks

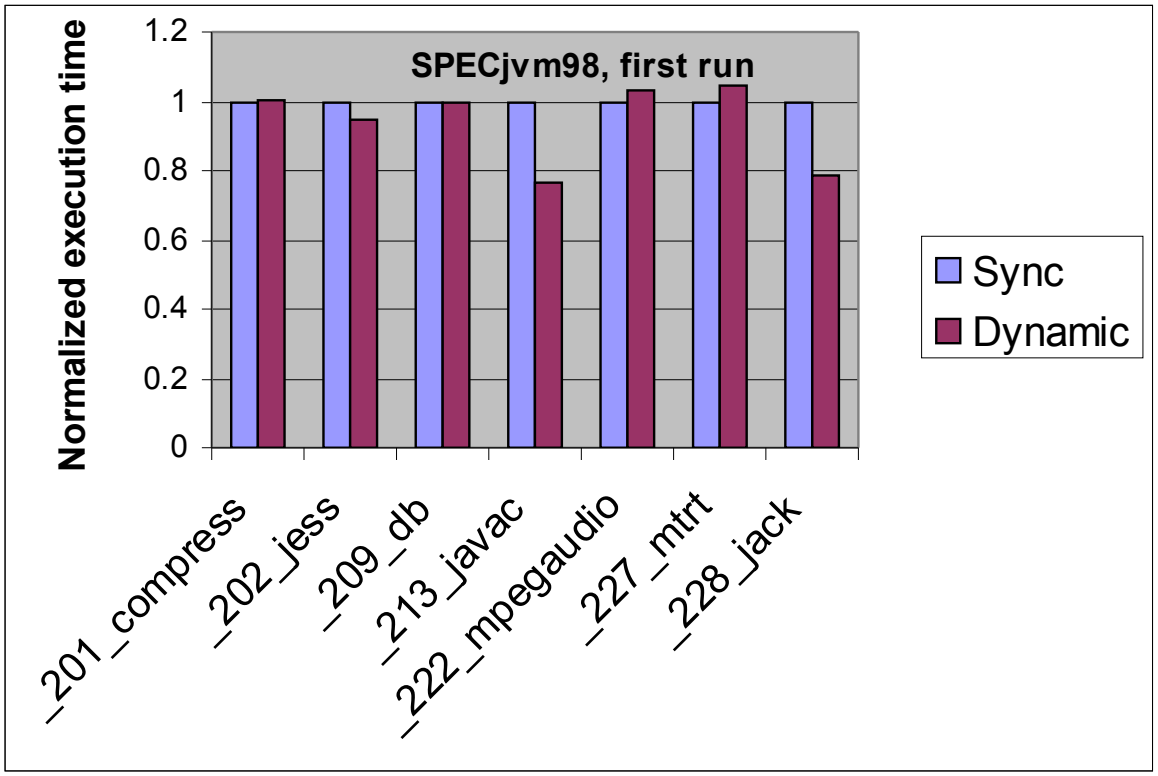
Limiting Negative Effect of Long Compilations

- Compilations may impede GC operation
  - GC requires exclusive VM access
  - Cannot allow class unloading while compilation in progress ➔ Compilations require VM access
- Solution
  - Compilation thread periodically releases and reacquires VM access allowing GC to cut-in
  - Upon re-acquiring VM access, check if GC unloaded any classes
  - If classes were unloaded, abort current compilation and retry

Ongoing Improvements to Asynchronous Compilation

- Idea
  - Use thread priorities to smooth out the effects of compilation - effectively interleave compilation with execution
- Implementation
  - Don’t use more than X% CPU for compilation
  - Use the queue of methods as a buffer
    - accumulate work during periods of heavy utilization
    - solve the backlog when CPU is lightly used or idle (due to IO for instance)
  - Prioritize compilation requests in the queue

Performance - Uniprocessors



Ahead Of Time (AOT) Compilation

- Using the JIT as a static compiler
- Fully compliant code
- Used by J2ME customers to decrease footprint by eliminating the JIT
- Will be used by the Real Time offering to eliminate the possibility of nondeterministic behaviour introduced by the JIT
- Experimenting with combining AOT and JIT compilation to improve startup times

Eclipse 3.0.1 Startup Times