OpenJDK Optimizations for AARCH64

Sean V Kelley, Ampere Computing
OpenJDK Benchmarking

● What is SpecJBB?
  ○ A usage model based on a world-wide supermarket company with an IT infrastructure that handles a mix of point-of-sale requests, online purchases and data-mining operations.
  ○ Both a pure throughput metric and a metric that measures critical throughput under service level agreements (SLAs) specifying response times ranging from 10ms to 100ms.
  ○ Support for multiple run configurations, enabling users to analyze and overcome bottlenecks at multiple layers of the system stack, including hardware, OS, JVM and application layers.

● How is it measured?
  ○ Max-JOPS – Throughput
  ○ Critical-JOPS – Throughput with latency requirements
How to approach

- Understand your performance gap on Java
  - HW
    - Is ARM Architecture worse than Intel? (weak vs. strong memory model, locks)
    - Is your SoC uArch worse than ARM? (degree of out of order, barriers)
  - SW
    - Is Java optimized for Intel better than ARM? (C2 code generation and optimization, GC)
    - Is Java optimized for ARM (Safepoints)
  - Configuration
    - Can we measure better? (SPECjbb2015 parameters, disk, memory, which GC,...)
Where to look in software

<table>
<thead>
<tr>
<th>Target Areas to Consider</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Garbage Collection</td>
<td>Critical jops is a good test for GC issues. Evaluate variation in heap and impact on performance. Compare GC models, e.g., parallelgc</td>
</tr>
<tr>
<td>Just in Time compiler</td>
<td>Can one achieve same max-jops performance without ‘best’ level of JIT optimization?</td>
</tr>
<tr>
<td>Safepoint sync</td>
<td>SW monitors can be used to find safepoint issues.</td>
</tr>
<tr>
<td>String Instructions</td>
<td>No performance change on x86 when instruction intrinsics not used</td>
</tr>
<tr>
<td>Encryption Instructions</td>
<td>Evaluate performance change when encryption turned off and when all transactions encrypted.</td>
</tr>
<tr>
<td>Feature</td>
<td>Comment</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>Understand topology and effect of noisy neighbors when shared</td>
</tr>
<tr>
<td>TLB</td>
<td>Heap size shrink decreased miss rate and measure performance. (Should see reduced performance)</td>
</tr>
<tr>
<td>Dependent loads block subsequent loads from issuing</td>
<td>Loop with dependent loads. What portion of the top functions profiled have this issue? Create micro-benchmarks.</td>
</tr>
<tr>
<td>Transparent Huge Pages / Page Size</td>
<td>Depends on workload</td>
</tr>
<tr>
<td>Memory Barriers</td>
<td>Use of data memory barriers in the absence of LSE atomics</td>
</tr>
<tr>
<td>Scaling with cores</td>
<td>Impact of topology and cache model on linear scaling with cores</td>
</tr>
</tbody>
</table>
Capturing the data

● Profiling
  ○ Take advantage of Perf
    ■ Combine with perf-map-agent, an open source agent supporting the JVM Tools Interface (JVMTI)
  ○ Make use of oprofile, especially its built-in support for JVMTI
  ○ Run jprofile/operf on JMH tests. Compare hot methods with x86.
  ○ Visualize your results with flamegraphs from the generated perf data
  ○ Use ARM’s DS5 StreamLine tool with it’s ability for live streaming and/or post-processing of your benchmark session.
    ■ Supports import of perf.data files
    ■ Includes range of kernel events, including ftrace support
Understanding the data

● Analyzing your results from a hardware perspective
  ○ Understanding your uARCH relative to ARMv8 supported features
    ■ Data memory barriers
    ■ Core topology and interconnect
  ○ Characterizing your uARCH features and their impact
    ■ L2 Cache model
      ● e.g., PMD model or per core
    ■ L3 Cache model
      ● e.g., victim cache
  ○ In comparative steps, applying equivalent penalty on x86
    ■ Inserting mfences where DMB would ordinarily be used on ARMv8
      ● Provides further insight into uARCH revisions needed
  ○ Generate mico-benchmarks from these tests to use in future uARCH models to keep a check on regressions, hopefully mitigated in those future designs.
Discussion
Thank you

- skelley@amperecomputing.com