15 Tools in 30 minutes

Paul Clarke, Software Architect / Developer
IBM

Join the Conversation #OpenPOWERSummit
Agenda

Advance Toolchain
XL C, C++, and Fortran compilers
FDPR
SDK for Linux on Power
SDK Migration Advisor
SDK Source Code Advisor
SDK CPI Breakdown & Drill-down
Power Functional Simulator
Performance Simulator (sim_ppc)
pipestat
LPCPU
pveclib
SPHDE
## Legend

<table>
<thead>
<tr>
<th>Requirement for best value</th>
<th>White box</th>
<th>Gray box</th>
<th>Black box</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source code changes</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source code access</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Recompile</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Object access</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Relink</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Advance Toolchain

- IBM Advance Toolchain for Linux on Power
- Latest open source compilers, runtime, and tools
- Enabled and optimized for the latest POWER processor
  - Quarterly updates, security fixes
- Supported
- Open source
Advance Toolchain

• Advance Toolchain 11.0-3
  – GCC 7.3.1
  – glibc 2.26 (libc, libm, libpthread, ...)
  – binutils 2.29
  – Boost 1.64
  – libdfp, PAFlib, gdb, Python, golang, GFortran, OProfile, Valgrind, itrace, TBB, Userspace RCU, libhugetlbfs, zlib, OpenSSL, TCMalloc, SPHDE, ...
## Advance Toolchain

<table>
<thead>
<tr>
<th>component</th>
<th>Red Hat Enterprise Linux 7.4</th>
<th>Red Hat Developer Toolset 7</th>
<th>Advance Toolchain 11.0-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>4.8.5</td>
<td>7.2.1</td>
<td>7.3.1</td>
</tr>
<tr>
<td>GNU libc (glibc)</td>
<td>2.17</td>
<td>-</td>
<td>2.26</td>
</tr>
<tr>
<td>binutils</td>
<td>2.25</td>
<td>2.28</td>
<td>2.29</td>
</tr>
<tr>
<td>Boost</td>
<td>1.53</td>
<td>-</td>
<td>1.64</td>
</tr>
<tr>
<td>runtime dependency?</td>
<td>-</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
XL C, C++, and Fortran compilers

- IBM's flagship compiler suite
- Proprietary
- Two editions:
  - Licensed, supported
  - Community Edition (*free, full-function*, unsupported)
- Used on AIX, z/OS, SPEC publishes
- New “clang” frontend for better source-code compatibility with GCC and LLVM and new language standards
XL C, C++, and Fortran compilers

- Automatic parallelization
- GPU acceleration
- OpenMP 3.1 and partial OpenMP 4.5
- High-performance libraries (MASS, BLAS)
- XML, HTML reports for further optimization insights
FDPR

- Feedback-Directed Program Restructuring
- IBM Research
- Post-link binary optimization tool
- Simple 3-step process: instrument, profile, optimize
- Gray box: requires preservation of relocation information at link-time
- Free download
$ gcc -o load -Wl,--emit-relocs load.c

$ /opt/ibm/fdprpro/bin/fdpr_instr_prof_opt load /home/pc/summit/load-2.1pc/load.instr
FDPR profiling: /home/pc/summit/load-2.1pc/load.instr ...
Total run time for 10000 iterations was: 72.699327 seconds

$ ./load
Total run time for 10000 iterations was: 8.755640 seconds

$ ./load.fdpr
Total run time for 10000 iterations was: 8.162184 seconds
FDPR

• FDPR journal: suggests source changes for performance improvement (white box)

$ gcc -o load -Wl,--emit-relocs -g load.c

$ /opt/ibm/fdprpro/bin/fdpr_instr_prof_jour load
/home/pc/summit/load-2.1pc/load.instr
FDPR profiling: /home/pc/summit/load-2.1pc/load.instr ...
Total run time for 10000 iterations was: 72.699327 seconds
$ ls -ltr | tail -1
load_jour.xml
High call overhead of a hot small function

Compiler: inline callee into caller - replace call to callee with its body

```
<operation name="Inline function">
  <problem>High call overhead of a hot small function</problem>
  <solution>Compiler: inline callee into caller - replace call to callee with its body</solution>
  <site>
    <ip>10000958</ip>
    <dir>/home/pc/summit/load-2.1pc</dir>
    <file>load.c</file>
    <fn>main</fn>
    <line>70</line>
    <xcount>265801472</xcount>
  </site>
  <param name="callee">
    <site>
      <ip>10000734</ip>
      <dir>/home/pc/summit/load-2.1pc</dir>
      <file>load.c</file>
      <fn>sum_add</fn>
      <line>18</line>
      <xcount>265801472</xcount>
    </site>
  </param>
</operation>
```
IBM SDK for Linux on Power

- Eclipse-based Integrated Development Environment (IDE)
- With IBM plug-ins for Linux on Power
- Runs on Linux x86 (or Linux VM on Windows or Mac) or Linux on Power
  - run IDE on Linux on Power, display back to laptop via remote X display, ssh X-Windows tunneling, or VNC
  - run entirely on x86, using cross-compilation and POWER emulation
  - run IDE on laptop, build and run on remote Power system
  - http://ibm.biz/ibmsdklop
    https://developer.ibm.com/linuxonpower/sdk
- Free download
- Videos: http://ibm.biz/linuxonpowervideos
IBM SDK for Linux on Power

Basic operation:

1. Launch SDK
2. Create new project, import code
3. Run Migration Advisor
4. Build (note Build Advisor output)
5. Run Source Code Advisor
6. Run perf or OProfile
7. Run CPI Breakdown & Drill-down
8. Run Power Performance Advisor
SDK: Migration Advisor

- SDK plug-in code-scanner
- Focus: Linux on x86 to Linux on Power
- Reports on portability issues and concerns
  - #ifdef x86
  - x86 syscalls, APIs, built-ins, assembly, pthread _np
  - long double, Float128
  - non-portable Hardware Transactional Memory
  - char type default signedness
  - 32/64 bit
  - sync built-ins
  - endian issues
- Simple: “Run Migration Advisor” from project's context menu
- Easy: Quick Fixes
- Efficient: Automated Quick Fix (!)
SDK: Migration Advisor
SDK: Migration Advisor

1 C/C++ files detected.
11 checkers enabled.
Analyzing load.c.
  5 x86-specific compiler built-in found.
    Fixed a possible issue at line 6.
    Fixed a possible issue at line 46.
    Fixed a possible issue at line 52.
    Fixed a possible issue at line 58.
Operation completed.
SDK: Source Code Advisor

- Run-time performance analysis and report suggested source code changes for improvement
- Leverages FDPR's journal
- Quick Fixes
SDK: Source Code Advisor

**INLINE FUNCTION**

- 100.00% in main()
- 50.01% on line 89 [/home/pc/test/load.c]
- 49.99% on line 93 [/home/pc/test/load.c]

**Problem:**
High overhead for frequent calls to a small function.

**Solution:**
Enable compiler optimization, if not already enabled. If optimization is enabled, tell the compiler to ignore some restrictions and inline the function: add the GNU extension "-ffunction-sections=always" to the declaration of the function to be inlined.
SDK: Source Code Advisor
SDK: CPI Breakdown & Drill-down

- Cycles-per-Instruction: a measure of processor efficiency
- On average, how many cycles does it take to execute each instruction? Lower is better!
- CPI Breakdown will breakdown CPI measurement into a hierarchy of processor-specific event classes
- Note: does not currently support POWER9
SDK: CPI Breakdown & Drill-down
SDK: CPI Breakdown & Drill-down
SDK: CPI Breakdown & Drill-down

- CPI Drill-down: profile based on specific hardware events
- Simple: double-click on events in breakdown view
  - A new profiling run is automatically launched
SDK: CPI Breakdown & Drill-down
Command-line tools from the SDK

- GUIs are powerful
- But, GUIs do not fit every customer, every situation
  - Difficult to automate
  - Expensive to develop and “get right”
  - Customer preference
- Command line tools for SDK features are now available
- All open source
- https://github.com/open-power-sdk
ma (migration advisor)

- Code scanner
- Focus: Linux on x86 to Linux on Power
- Reports on portability issues and concerns
  - list checkers here
- No “Quick Fixes”
- Open source (Python)
- https://github.com/open-power-sdk/migration-advisor
$ ma run ma/
[...]
================
Migration Report
================
Problem type: Non Portable Pthread
Problem description: Reports occurrences of non-portable Pthreads API
    File: ma/pthread.c
      Line: 3
      Problem: pthread_id_np_t tid

      Line: 4
      Problem: pthread_gettid_np()

Problem type: Performance degradation
Problem description: This preprocessor can contain code without Power optimization
    File: ma/performance.c
      Line: 3
      Problem: #ifdef _x86_
sca (source code advisor)

- Run-time performance analysis and report suggested source code changes for improvement
- Leverages FDPR's journal
- No Quick Fixes
- Open source (Python)
- [https://github.com/open-power-sdk/source-code-advisor](https://github.com/open-power-sdk/source-code-advisor)
$ sca load

[Problem: FIX LOAD-HIT-STORE]
[Description: A data store operation followed closely by a load from the same address causes the load to take extra time to complete.]
[Solution:
  1) Remove unnecessary "volatile" from variables.
  2) Use local variables instead of pointer references.
  Pseudo-example:
  loop { *p = foo(*p); }
  - Change to:
  p = *p
  loop { p = foo(p); }
  *p = p;
[Reference: /home/pc/load2/load.c:20 | Function: sum_add | Instruction Pointer: 10000774]
[Reference: /home/pc/load2/load.c:24 | Function: sum_sub | Instruction Pointer: 100007bc]

-------------------------------------------------------

[Problem: INLINE FUNCTION]
[Description: High overhead for frequent calls to a small function.]
[Solution:
  Enable compiler optimization, if not already enabled.
  If optimization is enabled, tell the compiler to ignore some restrictions and inline the function: add the GNU extension "__attribute__((always_inline))" to the declaration of the function to be inlined.
  Example: void __attribute__((always_inline)) foo(void);
  Note: Validate the results, as inlining can cause performance regressions in some scenarios.
[Reference: /home/pc/load2/load.c:88 | Function: main | Instruction Pointer: 100009d0]
[Reference: /home/pc/load2/load.c:92 | Function: main | Instruction Pointer: 10000a0c]
cpi (CPI breakdown)

- CPI Breakdown will breakdown CPI measurement into a hierarchy of processor-specific event classes
- 2-step process: record, display
- Open source (Python)
- https://github.com/open-power-sdk/cpi-breakdown
cpi (CPI breakdown)

$ cpi record ./load
  Recording CPI Events: 20/20 iterations (elapsed time: 168 seconds)
$ cpi display --file ./load_20180301_123123.cpi
[edited for brevity]
RUN_CPI: 1.398 (100.00 %)
  STALL_CPI: 0.989 (70.78 %)
    BRU_CRUSTALL_CPI: 0.001 (0.07 %)
    FXUSTALL_CPI: 0.218 (15.63 %)
    VSUSTALL_CPI: 0.000 (0.00 %)
    LSUSTALL_CPI: 0.610 (43.63 %)
    NTGCFUSH_CPI: 0.000 (0.00 %)
    NO_NTFSTALL_CPI: 0.000 (0.00 %)
    OTHERSTALL_CPI: 0.160 (11.45 %)
  NTCG_ALLFIN_CPI: 0.099 (7.09 %)
  THREAD_BLOCKSTALL_CPI: 0.019 (1.35 %)
    LWSYNCSTALL_CPI: 0.000 (0.00 %)
    HWSYNCSTALL_CPI: 0.000 (0.00 %)
    MEM_ECCDELAYSTALL_CPI: 0.000 (0.01 %)
    FLUSHSTALL_CPI: 0.019 (1.34 %)
    COQFULLSTALL_CPI: 0.000 (0.00 %)
    OTHERBLOCKSTALL_CPI: 0 (0.0 %)
  GCT_EMPTY_CPI: 0.012 (0.84 %)
    GCTEMPTY_ICMISS_CPI: 0.000 (0.03 %)
    GCTEMPTY_BRMPRED_CPI: 0.009 (0.67 %)
    GCTEMPTY_BRMPREDICMISS_CPI: 0.000 (0.00 %)
    GCTEMPTY_DISPHELD_CPI: 0.001 (0.06 %)
    GCTEMPTYOTHER_CPI: 0.001 (0.07 %)
  COMPLETION_CPI: 0.257 (18.40 %)
  OTHER_CPI: 0.022 (1.54 %)
$ cpi display --top-events 5 --top-metrics 5 --file ./load_20180301_123123.cpi

===============
Metrics Hot Spots
===============
  RUN_CPI : 1.398
  STALL_CPI : 0.989
  LSU_STALL_CPI : 0.610
  LSU_STALL_ST_FWD_CPI : 0.406
  COMPLETION_CPI : 0.257

===============
Events Hot Spots
===============
  PM_RUN_CYC : 37751459119
  PM_RUN_INST_CMPL : 27005146322
  PM_CMPLU_STALL : 26719546039
  PM_CMPLU_STALL_LSU : 16471141823
  PM_CMPLU_STALL_ST_FWD : 10959720878
curt

- System-wide utilization report
- Inspired by AIX “curt” command, but otherwise unrelated
- Full-system trace (“perf”) with specific events enabled
- Reports system-wide, per-process, per-task, per-CPU:
  - user time
  - kernel time
  - idle time
  - syscall time
  - HCALL time
  - HCALL time per syscall
  - interrupt time
  - task migrations
$ perf script -s ./curt.py

- PID:
  17096:
  
<table>
<thead>
<tr>
<th>task</th>
<th>command</th>
<th>cpu</th>
<th>user</th>
<th>sys</th>
<th>hv</th>
<th>busy</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>17096</td>
<td>curl</td>
<td>0</td>
<td>0.622258</td>
<td>0.455966</td>
<td>0.533538</td>
<td>0.000000</td>
<td>70.900980</td>
</tr>
<tr>
<td>17096</td>
<td>curl</td>
<td>7</td>
<td>4.957260</td>
<td>1.281960</td>
<td>0.638216</td>
<td>0.000000</td>
<td>184.465554</td>
</tr>
<tr>
<td>17096</td>
<td>curl</td>
<td>ALL</td>
<td>5.579518</td>
<td>1.737926</td>
<td>1.171754</td>
<td>0.000000</td>
<td>255.366534</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>task</th>
<th>command</th>
<th>cpu</th>
<th>runtime</th>
<th>sleep</th>
<th>wait</th>
<th>blocked</th>
<th>iowait</th>
<th>unaccounted</th>
<th>util%</th>
<th>moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>17096</td>
<td>curl</td>
<td>0</td>
<td>2.215550</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.084866</td>
<td></td>
</tr>
<tr>
<td>17096</td>
<td>curl</td>
<td>7</td>
<td>7.285182</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.008204</td>
<td></td>
</tr>
<tr>
<td>17096</td>
<td>curl</td>
<td>ALL</td>
<td>9.500732</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.084866</td>
<td>3.0%</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>count</th>
<th>elapsed</th>
<th>pending</th>
<th>average</th>
<th>minimum</th>
<th>maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>mmap</td>
<td>72</td>
<td>0.382168</td>
<td>0.000000</td>
<td>0.005308</td>
<td>0.002336</td>
<td>0.027584</td>
</tr>
<tr>
<td>close</td>
<td>38</td>
<td>0.028054</td>
<td>0.000000</td>
<td>0.000738</td>
<td>0.000510</td>
<td>0.003484</td>
</tr>
<tr>
<td>mprotect</td>
<td>37</td>
<td>0.205024</td>
<td>0.000000</td>
<td>0.005541</td>
<td>0.001896</td>
<td>0.008204</td>
</tr>
<tr>
<td>open</td>
<td>36</td>
<td>0.181678</td>
<td>0.000000</td>
<td>0.005047</td>
<td>0.003246</td>
<td>0.016274</td>
</tr>
<tr>
<td>fstat</td>
<td>35</td>
<td>0.029962</td>
<td>0.000000</td>
<td>0.000556</td>
<td>0.000366</td>
<td>0.003978</td>
</tr>
<tr>
<td>read</td>
<td>33</td>
<td>0.109874</td>
<td>0.000000</td>
<td>0.003330</td>
<td>0.000718</td>
<td>0.030738</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(hvc)</th>
<th>name</th>
<th>count</th>
<th>elapsed</th>
<th>pending</th>
<th>average</th>
<th>minimum</th>
<th>maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 8)H_ENTER</td>
<td>249</td>
<td>0.471600</td>
<td>0.000000</td>
<td>0.001894</td>
<td>0.001136</td>
<td>0.006916</td>
<td></td>
</tr>
<tr>
<td>( 4)H_REMOVE</td>
<td>152</td>
<td>0.260904</td>
<td>0.000000</td>
<td>0.001716</td>
<td>0.001120</td>
<td>0.002862</td>
<td></td>
</tr>
<tr>
<td>(292)H_BULKREMOVE</td>
<td>39</td>
<td>0.102006</td>
<td>0.000000</td>
<td>0.002616</td>
<td>0.001366</td>
<td>0.007068</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(irq)</th>
<th>count</th>
<th>elapsed</th>
<th>pending</th>
<th>average</th>
<th>minimum</th>
<th>maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 20)</td>
<td>1</td>
<td>0.051650</td>
<td>0.000000</td>
<td>0.051650</td>
<td>0.051650</td>
<td>0.051650</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>task</th>
<th>command</th>
<th>cpu</th>
<th>user</th>
<th>sys</th>
<th>hv</th>
<th>busy</th>
<th>idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>ALL</td>
<td>6.164324</td>
<td>2.401406</td>
<td>1.456634</td>
<td>0.000000</td>
<td>512.169142</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>curl</td>
<td>10.985284</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>202.917200</td>
<td>1.0%</td>
</tr>
</tbody>
</table>
### CPU Usage

<table>
<thead>
<tr>
<th>Task</th>
<th>CPU Usage</th>
<th>User Time</th>
<th>System Time</th>
<th>HV Time</th>
<th>Busy Time</th>
<th>Idle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>81.803774</td>
<td>191.490346</td>
<td>1.541632</td>
<td>0.461220</td>
<td>2629.644696</td>
<td></td>
</tr>
</tbody>
</table>

### Task Breakdown

<table>
<thead>
<tr>
<th>Task</th>
<th>Runtime</th>
<th>Sleep</th>
<th>Wait</th>
<th>Blocked</th>
<th>IOWait</th>
<th>Unaccounted</th>
<th>Util%</th>
<th>Moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>266.336282</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1620.639964</td>
<td>9.0%</td>
<td>5</td>
</tr>
</tbody>
</table>

### Memory Operations

<table>
<thead>
<tr>
<th>Name</th>
<th>Count</th>
<th>Elapsed</th>
<th>Pending</th>
<th>Average</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>134926</td>
<td>189.079072</td>
<td>0.000000</td>
<td>0.001401</td>
<td>0.001194</td>
<td>0.159456</td>
</tr>
<tr>
<td>Mmap</td>
<td>83</td>
<td>0.417228</td>
<td>0.000000</td>
<td>0.005026</td>
<td>0.001994</td>
<td>0.027584</td>
</tr>
<tr>
<td>Close</td>
<td>51</td>
<td>0.037880</td>
<td>0.000000</td>
<td>0.000742</td>
<td>0.000482</td>
<td>0.003484</td>
</tr>
<tr>
<td>Open</td>
<td>45</td>
<td>0.214196</td>
<td>0.000000</td>
<td>0.004759</td>
<td>0.002254</td>
<td>0.016274</td>
</tr>
<tr>
<td>Read</td>
<td>45</td>
<td>0.175362</td>
<td>0.054250</td>
<td>0.003896</td>
<td>0.000718</td>
<td>0.030738</td>
</tr>
</tbody>
</table>

### High-Voltage Control

<table>
<thead>
<tr>
<th>HVC</th>
<th>Count</th>
<th>Elapsed</th>
<th>Pending</th>
<th>Average</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>H_ENTER</td>
<td>329</td>
<td>0.678906</td>
<td>0.000000</td>
<td>0.002063</td>
<td>0.001112</td>
<td>0.032556</td>
</tr>
<tr>
<td>H_REMOVE</td>
<td>190</td>
<td>0.314528</td>
<td>0.000000</td>
<td>0.001655</td>
<td>0.001094</td>
<td>0.002862</td>
</tr>
<tr>
<td>H_BULK_REMOVE</td>
<td>48</td>
<td>0.128092</td>
<td>0.000000</td>
<td>0.002668</td>
<td>0.001366</td>
<td>0.007068</td>
</tr>
<tr>
<td>H_RANDOM</td>
<td>7</td>
<td>0.006144</td>
<td>0.000000</td>
<td>0.000877</td>
<td>0.000604</td>
<td>0.001808</td>
</tr>
<tr>
<td>H_PROTECT</td>
<td>6</td>
<td>0.007320</td>
<td>0.000000</td>
<td>0.001220</td>
<td>0.000770</td>
<td>0.002390</td>
</tr>
</tbody>
</table>

### Interrupts

<table>
<thead>
<tr>
<th>Name</th>
<th>Count</th>
<th>Elapsed</th>
<th>Pending</th>
<th>Average</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>H_ENTER</td>
<td>329</td>
<td>0.678906</td>
<td>0.000000</td>
<td>0.002063</td>
<td>0.001112</td>
<td>0.032556</td>
</tr>
<tr>
<td>H_REMOVE</td>
<td>190</td>
<td>0.314528</td>
<td>0.000000</td>
<td>0.001655</td>
<td>0.001094</td>
<td>0.002862</td>
</tr>
<tr>
<td>H_BULK_REMOVE</td>
<td>48</td>
<td>0.128092</td>
<td>0.000000</td>
<td>0.002668</td>
<td>0.001366</td>
<td>0.007068</td>
</tr>
<tr>
<td>H_RANDOM</td>
<td>7</td>
<td>0.006144</td>
<td>0.000000</td>
<td>0.000877</td>
<td>0.000604</td>
<td>0.001808</td>
</tr>
<tr>
<td>H_PROTECT</td>
<td>6</td>
<td>0.007320</td>
<td>0.000000</td>
<td>0.001220</td>
<td>0.000770</td>
<td>0.002390</td>
</tr>
</tbody>
</table>

Total Trace Time: 264.750018 ms
Power Functional Simulator

- Full-system Power hardware simulator
- Access a Power System without access to a Power System!
- Great for basic Linux on Power application development, porting
- Single core, single thread
- Ease-of-use wrappers at https://github.com/open-power-sdk/power-simulator
You are starting the IBM POWER9 Functional Simulator
When the boot process is complete, use the following credentials to access it via ssh:

```
ssh root@172.19.98.109
password: mambo
```

Licensed Materials - Property of IBM.
(C) Copyright IBM Corporation 2001, 2017
All Rights Reserved.
Using initial run script /opt/ibm/systemsim-p9/run/p9/linux/boot-linux-le-skiboot.tcl
Found skiboot skiboot lids in current directory
Found kernel vmlinux in current directory
Booting with skiboot ./skiboot.lid...
Booting with kernel ./vmlinux...
INFO: 0: (0): !!!!!!! Simulator now in TURBO mode !!!!!!!
OPAL v5.7-107-g8fb78ae starting...
[...]
Linux version 4.13.0-rc4+ (pc@moose1.pok.stglabs.ibm.com) (gcc version 4.8.5 20150623 (Red Hat 4.8.5-11) (GCC)) #2 SMP Fri Aug 18 17:01:57 EDT 2017
[...]
Debian GNU/Linux 9 mambo ppc64le 172.19.98.109
mambo login:
Performance Simulator (sim_ppc)

- Cycle-accurate POWER processor simulator
- Transforms instruction traces into processor cycle reports
- Cycle reports can be viewed using included viewers (ScrollPipeViewer and jviewer)
- Use Valgrind itrace to record instruction trace
  - Not part of standard Valgrind
  - Included in Advance Toolchain's Valgrind
Performance Simulator (sim_ppc)

Basic usage scenario:

1. Record instruction trace (.vgi file)
   
   $ valgrind --tool=itrace --binary-outfile=tracefile.vgi --num-K-insns-to-collect=100 --demangle=no /bin/ls

2. Convert .vgi to .qt
   
   $ vgi2qt -f tracefile.vgi -o tracefile.qt

3. Run sim_ppc cycle-accurate timer (.pipe file)
   
   $ /opt/ibm/sim_ppc/sim_p8/bin/run_timer tracefile.qt 100000 10000 1 tracefile -scroll_pipe 1 -scroll_begin 1 -scroll_end 100000

4. Run viewer
   
   $ /opt/ibm/sim_ppc/bin/scrollpv tracefile.pipe
Performance Simulator (scrollpv)
Performance Simulator (jviewer)
pipestat

- Automated analysis of processor instruction cycle reports (from Performance Simulator)
  - Most executed loops
  - Most executed misaligned short loops
  - Most executed blocks with long latency instructions
  - Most executed blocks with redundant loads
  - Most executed incorrectly hinted branches
  - Most executed mispredicted branches
  - Most frequently mispredicted branches
  - Most executed code paths that have a store followed soon after by a load of the same address
  - Most load-hit-store related events on a particular instruction address
  - Most executed instructions where the result is used a small number of instructions later but takes a large number of cycles before the dependent instruction starts
pipestat

$ pipestat tracefile.qt          # [heavily edited for brevity, clarity]
HOT execution count blocks:
0x000000004025c98-0x000000004025cac N:678 6 inst trace inst 297
0x000000004025dc8-0x000000004025dd8 N:784 5 inst trace inst 923
0x00000000400df48-0x0000000040df7c N:248 14 inst trace inst 454

HOT misaligned short loops:
0x0000000040ea58-0x0000000040ea74 N:170 8 inst short misalign32

Loop size summary data:
Instructions loops total_iter min_iter max_iter avg_iter total_inst % of trace
6 1 402 402 402 402.00 2412 2.93
8 1 170 170 170 170.00 1360 1.65

HOT Loop constructs (5 total)

HOT long latency instruction blocks:
0x0000000040ea7c-0x0000000040ea90 N:48 6 inst badness 240
0x0000000040e7d0-0x0000000040e7ec N:55 8 inst badness 206
0x0000000040173f0-0x00000000401748 N:235 7 inst badness 168
pipestat

HOT redundant loads: intra+stack: 261 (0.32%) inter+stack: 948 (1.15%) intra: 0 (0.00%) inter: 0 (0.00%)
0x00000400e0d4-0x00000400e0f4 N:200 redundant loads 600
0x0000040173f0-0x000004017408 N:235 redundant loads 235
0x00000400e2ec-0x00000400e2f8 N:235 redundant loads 230

HOT bad branch hints:
0x000004025dc4 hint likely not taken but was taken 36.47% (450/1234)
0x000004025db0 hint likely not taken but was taken 15.84% (122/770)
0x00000400e124 hint likely taken but was not taken 100.00% (87/87)

HOT branches with high linear branch entropy and executed frequently
0x00000400e824 N:110      LBE 0.00000 0.6909/0.0000
0x00000400e338 N:48       LBE 0.00000 0.5417/0.0000
0x00000400e90c N:200      LBE 0.14444 0.3600/0.1444

HOT load hit store separated by less than 100 instructions:
Store IA Load IA Count % count min count Avg max Dev. loads store registers AGEN Store Values
00000400df64 00000400e0dc 200 100.0% 200 13 87 34.17 68 358.1 200 1/1/1 st: G1 ld: G1
00000400ebac 00000400e1b0 48 100.0% 48 37 48 37.00 37 0.0 0 22/22/22 st: G1 ld: G1 BASE REG CHANGED: 48
00000400eba8 00000400e1ac 48 100.0% 48 37 48 37.00 37 0.0 0 23/23/23 st: G1 ld: G1 BASE REG CHANGED: 48

Total LHS events: 3282 15.1% of loads 40.0% of stores
Inst annotated disassembly

00000400df48 > mflr r0 N:248 HOT-blk Use:7 HOT-LongLat LBAD 5 from:0x400e858,0x400e928
00000400df4c M std r30,-16(r1) N:248 HOT-blk HOT-LHS DA ref to intra 0x0400e0e8 22i 35.08%
00000400df50 M std r31,-8(r1) N:248 HOT-blk HOT-LHS DA ref to intra 0x0400e0ec 22i 35.08%
00000400df54 mr r30,r3,r3 N:248 HOT-blk Use:11
00000400df58 M std r28,-32(r1) N:248 HOT-blk HOT-LHS DA ref to intra 0x0400e0e0 17i 35.08%
00000400df5c M std r29,-24(r1) N:248 HOT-blk HOT-LHS DA ref to intra 0x0400e0e4 17i 35.08%
00000400df60 mr r31,r11,r11 N:248 HOT-blk Use:9
00000400df64 std r0,16(r1) N:248 HOT-blk HOT-LHS DA ref to intra 0x0400e0dc 13i 35.08%
00000400df68 std u r1,-64(r1) N:248 HOT-blk Use:63
00000400df6c ld r10,8(r3) N:248 HOT-blk Use:2
00000400df70 lbz r9,4(r3) N:248 HOT-blk Use:2
00000400df74 cmpdi cr7,r10,0 N:248 HOT-blk Use:2
00000400df78 clrldi r8,r9,60 N:248 HOT-blk Use:7.2
00000400df7c C beq- cr7,0x400e120 N:248 HOT-blk Hint:unlikely HOT BAD HINT HOT-LBE to:0x400e120 br prob:35.08% (avg seq tk 1.2 ntk 2.3 LBE 0.079 0.169/0.079)

Opcode Mix

<table>
<thead>
<tr>
<th>Opcode Name</th>
<th>Mask</th>
<th>Match</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld</td>
<td>fc000003</td>
<td>e8000000</td>
<td>11654</td>
</tr>
<tr>
<td>std</td>
<td>fc000003</td>
<td>f8000000</td>
<td>6560</td>
</tr>
<tr>
<td>cmpi</td>
<td>fc200000</td>
<td>2c200000</td>
<td>5255</td>
</tr>
<tr>
<td>bc</td>
<td>ffc00003</td>
<td>40c00000</td>
<td>4000</td>
</tr>
</tbody>
</table>

Total loads: 21672 Total stores: 8203
GPR: loads: 21672 stores: 8203
pveclib

- Header files that contain well crafted implementations of useful vector and large number operations the Power ISA Vector Facilities
- Provide higher order functions not provided directly by the PowerISA
- Open source
- https://github.com/open-power-sdk/pveclib
pveclib

- udiv_qrnn
- fxu_bcdadd, fxu_bcd_sub
- vec_BCD2DFP, vec_DFP2BCD
- vec_bcdadd, vec_bcdsub, vec_bcdmul, vec_bcddiv
- vec_shift_leftdo
- vec_isalpha, vec_isalnum, vec_isdigit
- vec_toupper, vec_tolower
- vec_absdub
- vec_revq, vec_revd, vec_revw, vec_revh
- vec_clzq, vec_popcntq
- vec_sldq, vec_srqi, vec_srq, vec_slqi, vec_slq
- vec_pasted
- vec_mulouw, vec_muleuw, vec_mulosw, vec_mulesw
- vec_adduqm, ...
- ...
SPHDE

- Shared Persistent Heap Data Environment
  - Shared Address Space (SAS)
  - Shared Persistent Heap
  - Cross-platform
  - Lockless
- Shared memory using a shared heap allows passing valid pointers between processes, easing implementation of multiprocessing
- Lockless multiprocess logger
- Lockless producer-consumer queue (fast IPC)
- Fast timestamps
- Open source
- https://github.com/sphde/sphde
LPCPU

- Linux Performance Customer Profiler Utility
- Collects a large, customizable set of system information and performance data for offline analysis
- Two-step process:
  - Collect data (produces a compressed tar file)
  - Postprocess data (collates data, produces interactive HTML)
- Open source
- [http://ibm.co/download-lpcpu](http://ibm.co/download-lpcpu)
LPCPU

- iostat
- mpstat
- vmstat
- perf or OProfile
- meminfo
- top
- sar
- /proc/interrupts
- tcpdump
- kernel trace
- hardware performance counters
- netstat
- ...

...
# tar -jxf lpcpu.tar.bz2
# cd lpcpu
# ./lpcpu.sh duration=15 extra_profilers="perf tcpdump"
[…]
Packaging data...data collected is in
/tmp/lpcpu_data.hostname.default.2018-02-26_1000.tar.bz2
[optionally, copy elsewhere...]
# tar -xf ./lpcpu_data.hostname.default.2018-02-26_1000.tar.bz2
# cd lpcpu_data.hostname.default.2018-02-26_1000
# ./postprocess.sh

- Then point a browser at the summary.html file
LPCPU
Support

- Formal Support (Advance Toolchain, XL compilers)
- Linux on Power Developer Portal
  - (replaces developerWorks Linux on Power Community)
- StackOverflow, ServerFault, StackExchange, SuperUser, ...
- IRC channels (FreeNode #ubuntu-powerpc, #fedora-ppc, ...)
Thank You!