Tools for applications improvement

George Bosilca

Innovative Computing Laboratory
COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF TENNESSEE
Performance vs. Correctness

» MPI is a corner-stone library for most of the HPC applications
» Delivering high performance require a fast and reliable MPI library
» It require a well designed application
» And well tested libraries ...
» Describe a fine grained execution of the application
» The applications is split in atomic blocks and information is gathered about each of the blocks.
  » How often each line of code is executed
  » Which lines are actually executed
  » How long each block take
GNU coverage tool

- Allow to write tests that cover all the possible execution path
- Give information about the most expensive parts/blocks of the code
- Is this a very useful information ?!
- Require a exhaustive knowledge of the application architecture to be able to provide useful information
GNU profiling tool

- Provide information about:
  - How many times each function has been executed
  - How percentage of the total time has been spent inside each function
  - Dependencies between functions
  - Call-graph of the execution

- Each sample counts as 0.01 seconds.

<table>
<thead>
<tr>
<th>% cumulative</th>
<th>cumulative</th>
<th>self</th>
<th>self</th>
<th>total time</th>
<th>seconds</th>
<th>seconds</th>
<th>calls</th>
<th>us/call</th>
<th>us/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.00</td>
<td>0.01</td>
<td>0.01</td>
<td>2352</td>
<td>4.25</td>
<td>4.25</td>
<td>mca_btl_mvapi_component_progress</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>2134</td>
<td>0.00</td>
<td>0.00</td>
<td>ompi_group_free</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>1920</td>
<td>0.00</td>
<td>0.00</td>
<td>ompi_errcode_get_mpi_code</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>1710</td>
<td>0.00</td>
<td>0.00</td>
<td>right_rotate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>1672</td>
<td>0.00</td>
<td>0.00</td>
<td>mca_mpool_sm_free</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>1293</td>
<td>0.00</td>
<td>0.00</td>
<td>mca_coll_sm_allreduce_intra</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Valgrind

» Suite of tools for debugging and profiling Linux programs.
  » Memcheck: detect erroneous memory accesses
    » Use of uninitialized memory
    » Reading/writing memory after it has been free’d
    » Reading/writing off the end of malloc’d blocks
    » Reading/writing inappropriate areas on the stack
    » Memory leaks
    » Passing of uninitialized and/or unaddressible memory to system calls
    » Mismatched use of malloc/new/new[] vs. free/delete/delete[]
Valgrind

Sysexec param writev(vector[...]) contains uninitialised or unaddressable byte(s)

at 0x3FDBDB: writev (in /lib/tls/libc-2.3.4.so)
by 0x1BBF662: mca_btl_tcp_frag_send (btl_tcp_frag.c:78)
by 0x1BBF9765: mca_btl_tcp_endpoint_send (btl_tcp_endpoint.c:224)
by 0x1BBF6881: mca_btl_tcp_send (btl_tcp.c:394)
by 0x1BBC9021: mca_bml_base_send (bml.h:246)
by 0x1BBC8AD6: mca_pml obl_recv_request_ack (pml obl_recvreq.c:283)
by 0x1BBC9BF2: mca_pml obl_recv_request_progress (pml obl_recvreq.c:508)
by 0x1BBCA744: mca_pml obl_recv_request_match_specific (pml obl_recvreq.c:807)
by 0x1BBC2B98: mca_pml obl irecv (pml obl irecv.c:65)
by 0x1BC189B2: omni_coll_tuned_sendrecv_actual_localcompleted (coll_tuned_util.c:81)
Address 0x80A42FA is not stack'd, malloc'd or (recently) free'd

ERROR SUMMARY: 1974 errors from 2 contexts (suppressed: 122 from 2)
malloclfree: in use at exit: 0 bytes in 0 blocks.
malloclfree: 0 allocs, 0 frees, 0 bytes allocated.
For a detailed leak analysis, rerun with: --leak-check=yes
For counts of detected errors, rerun with: -v
[bosilca@thor1 VEX]$
Valgrind

» Cachegrind
  » Cache profiler
  » Detailed simulation of all levels of cache
  » Accurately pinpoint the source of cache misses in the code.
  » Statistics: number of cache misses, memory references and instructions executed for each line of code, function, module and program.
Valgrind

» Callgrind
   » It’s more than just profiling !!!
   » Automatically instrument at runtime the application and gather information about the call-graphs and the timings
   » A visual version of gprof
### mca_pml_ob1_recv

<table>
<thead>
<tr>
<th>Incl.</th>
<th>Distance</th>
<th>Called</th>
<th>Caller</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.00</td>
<td>6-7 (6)</td>
<td>3</td>
<td>0x003257C0 (ld-2.3.4.so)</td>
</tr>
<tr>
<td>100.00</td>
<td>5-6 (5)</td>
<td>3</td>
<td>0x0804916C (NPmpi)</td>
</tr>
<tr>
<td>100.00</td>
<td>4-5 (4)</td>
<td>3</td>
<td>_libc_start_main (libc-2.3.4.so)</td>
</tr>
<tr>
<td>100.00</td>
<td>3-4 (3)</td>
<td>3</td>
<td>659</td>
</tr>
<tr>
<td>95.69</td>
<td>1</td>
<td>3</td>
<td>593</td>
</tr>
<tr>
<td>95.62</td>
<td>2</td>
<td>3</td>
<td>577</td>
</tr>
<tr>
<td>4.31</td>
<td>3</td>
<td>66</td>
<td>Sync (NPmpi: mpi.c)</td>
</tr>
<tr>
<td>4.31</td>
<td>2</td>
<td>66</td>
<td>MPI_Barrier (libmpi.so.0.0.0: pbarrier.c)</td>
</tr>
<tr>
<td>4.31</td>
<td>1</td>
<td>66</td>
<td>mca_coll_basic_barrier_intra_lin (mca_coll_basic.s...)</td>
</tr>
<tr>
<td>0.07</td>
<td>2</td>
<td>16</td>
<td>RecvRepeat (NPmpi: mpi.c)</td>
</tr>
</tbody>
</table>

---

**# | Ir | Assembler | Source Position**
---

1. There is no instruction info in the profile data file.
2. For the Valgrind Calltree Skin, rerun with option --dump-instr=yes
3. To see (conditional) jumps, additionally specify --trace-jump=yes
The power of the 2P

» Further improvement of the MPI library require an detailed understanding of the lifetime cycle of the point-to-point operations

» There are 2 parts:
  » All local overheads on the sender and the receiver
  » And the network overhead/latency

» However, we cannot improve the network latency ...
Peruse

» A standard to-be?
» An MPI extension for revealing unexposed implementation information...
» Similar to PAPI (who expose processor counters) but exposing MPI request events
» PERUSE is: A set of events tracing the lifetime of an MPI request
PAPI

» Use the processor counters to give information about the execution
  » Only few of them are interesting for the MPI library
    » Cache misses
    » Instruction counters
    » TLB
  » Cache disturbances
The 2P

» Mixing PERUSE and PAPI
» At each step in the lifetime of a request we gather:
  » Cache misses
  » Instruction counter
» Therefore we can compute accurately the cost of each of the steps
The 2P

» PERUSE will be included in the main Open MPI trunk shortly
» The PAPI extension is in a quite advanced state
» This is still work in progress ...
Does it really work?
Does it really work?
Conclusion

» Performance always start from the first design step
» A bad design decision have a persistent impact on the overall performance of the application
» Not always easy to achieve especially when several peoples are involved
» But there is hope:
  » Now we have the tools required to work on
Open MPI multi-rail

» Actual algorithm depend only on the priority
» Similar to collectives we plan to use a model to predict the
  » Message size from where multi-rail make sense
  » Determine the size of each segment based on the latency and the bandwidth of the network.