Binary Instrumentation Support for Measuring Performance in OpenMP Programs

Mustafa Elfituri   Jeanine Cook   Jonathan Cook
New Mexico State University

SECSE 2013 @ ICSE 2013
May 18, 2013
double findSubGraphs(graph* G, 
    edge* maxIntWtList, int maxIntWtListSize)
{
    ...
    #pragma omp parallel
    {
        #pragma omp barrier
        ...
        #pragma omp for
            for (vert=start[phase_num];
                 vert<start[phase_num+1]; vert++) {
                ...
                int myLock = omp_test_lock(&vLock[w]);
                if (myLock) {
                ...
OpenMP Tools

- Making common tools for OpenMP is hard
  - Source level standard does not include monitoring standard
  - E.g., MPI has the PMPI interception standard
- Commercial compilers have their own private OpenMP tools
- Opari2 is the only active open tool
  - Uses source translation techniques
Source Translation is Tricky!

- Harder to fit into a development toolchain
- Source code in real applications can get very complicated!
- Modern programming languages are not toy LALR(1) grammars!
- Tool effort can bog down in managing source instrumentation issues
- Commercial compiler OpenMP tools use binary instrumentation
Example: Intel Threading Tools

“Binary Instrumentation for Intel Thread Profiler works better with the OpenMP* Compatibility Libraries (dynamic version: libiomp5.so or libguide40.so) available via an Intel Compiler. This library has been instrumented for Intel Thread Profiler with the User-Level Synchronization API's. This library is used by default with the Intel Compiler, and can be used with an OpenMP* GCC* compiled application. If a 3rd party OpenMP* library is used, Thread Profiler can still collect data, but Intel Thread Profiler will not comprehend the OpenMP calls - it will be analyzed as a POSIX* application.”

Example: IBM's OpenMP

“DPOMP is developed based on IBM’s dynamic instrumentation infrastructure (DPCL). This supports binary instrumentation of FORTRAN, C and C++ programs. The DPOMP Tool was developed for dynamic instrumentation of OpenMP applications. It inserts into the application binary calls to a POMP (Performance Monitoring Interface for OpenMP) compliant library. The DPOMP tool reads the binary of the application, as well as the binary of a POMP compliant library and instruments the binary of the application with calls defined in the POMP compliant library. DPOMP requires DPCL version 3.2.6.”

“The POMP OpenMP Performance Monitoring Interface is a proposed API for enabling programmers and performance tools to obtain information about the performance of OpenMP constructs in an OpenMP program. The IBM compilers and HPCT toolkit provide a prototype implementation of some of the POMP functionality. The full POMP API provides a number of events to report the time spent in different parts of compiler-instrumented user code, and the prototype POMP implementation provides a core subset of the events, sufficient to instrument most OpenMP programs. The current POMP implementation allows profiling of Parallel Regions, WorkShare Do and Parallel Do Loops.”

https://www.alcf.anl.gov/user-guides/bgp-pomp
Gnu OpenMP

OpenMP Program

libGOMP Runtime
OpenMP Parallel Section

int main()
{
    ...
    #pragma omp parallel ...
    {
    ...
    }
    ...
}

8048714: call 8048570 <GOMP_parallel_start@plt
8048719: lea  0x14(%esp),%eax
804871D: mov  %eax,(%esp)
8048720: call 8048796 <main._omp_fn.0>
8048725: call 8048590 <GOMP_parallel_end@plt>
OpenMP Parallel For

#pragma omp parallel ...
{
    #pragma omp for ...
    for (i=0; I < 100000; ++i)
    { ... }
}

...
80487F9: cmp    %edx,-0x10(%ebp)
8048800: jl     80487f5 <main._omp_fn.0+0x5f>
8048802: call   8048580 <GOMP_barrier@plt>
OpenMP Critical Section

```c
#pragma omp parallel ...
{
    #pragma omp critical
    { ... }
}

8048807: call 8048620 <GOMP_critical_start@plt
...
8048855: call 80485b0 <GOMP_critical_end@plt>
```
PGOMP Profiling Interception

- OpenMP Program
- PGOMP Interception
- libGOMP Runtime
<table>
<thead>
<tr>
<th>Functions Intercepted by PGOMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOMP_parallel_start</td>
</tr>
<tr>
<td>GOMP_parallel_end</td>
</tr>
<tr>
<td>GOMP_barrier</td>
</tr>
<tr>
<td>GOMP_critical_start</td>
</tr>
<tr>
<td>GOMP_critical_end</td>
</tr>
<tr>
<td>GOMP_critical_name_start</td>
</tr>
<tr>
<td>GOMP_critical_name_end</td>
</tr>
<tr>
<td>GOMP_single_start</td>
</tr>
<tr>
<td>omp_init_lock</td>
</tr>
<tr>
<td>omp_destroy_lock</td>
</tr>
<tr>
<td>omp_set_lock</td>
</tr>
<tr>
<td>omp_test_lock</td>
</tr>
<tr>
<td>omp_unset_lock</td>
</tr>
<tr>
<td>omp_set_nest_lock</td>
</tr>
<tr>
<td>omp_test_nest_lock</td>
</tr>
<tr>
<td>omp_unset_nest_lock</td>
</tr>
</tbody>
</table>
## PGOMP Trace Mode

<table>
<thead>
<tr>
<th>Name</th>
<th>Return-address</th>
<th>ThreadID</th>
<th>EnterTime</th>
<th>ExitTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOMP_barrier</td>
<td>0x8049875</td>
<td>0</td>
<td>0.030259</td>
<td>0.030260</td>
</tr>
<tr>
<td>GOMP_parallel_end</td>
<td>0x8049ab8</td>
<td>0</td>
<td>0.030265</td>
<td>0.030268</td>
</tr>
<tr>
<td>GOMP_parallel_start</td>
<td>0x804a5b6</td>
<td>0</td>
<td>0.030320</td>
<td>0.030399</td>
</tr>
<tr>
<td>GOMP_barrier</td>
<td>0x804a1a6</td>
<td>3</td>
<td>0.030400</td>
<td>0.030408</td>
</tr>
<tr>
<td>GOMP_barrier</td>
<td>0x804a1a6</td>
<td>0</td>
<td>0.030407</td>
<td>0.030408</td>
</tr>
<tr>
<td>GOMP_barrier</td>
<td>0x804a1a6</td>
<td>2</td>
<td>0.030399</td>
<td>0.030408</td>
</tr>
<tr>
<td>GOMP_barrier</td>
<td>0x804a1a6</td>
<td>1</td>
<td>0.030399</td>
<td>0.030408</td>
</tr>
<tr>
<td>omp_set_lock</td>
<td>0x804a28b</td>
<td>3</td>
<td>0.030492</td>
<td>0.030492</td>
</tr>
<tr>
<td>omp_unset_lock</td>
<td>0x804a2ab</td>
<td>3</td>
<td>0.030497</td>
<td>0.030497</td>
</tr>
<tr>
<td>Name</td>
<td>StartAddress</td>
<td>EndAddress</td>
<td>ThreadID</td>
<td>WaitTime</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------</td>
<td>------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>GOMP_parallel_start</td>
<td>0x804bee4</td>
<td>0x804bef1</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>omp_test_lock</td>
<td>0x804b92e</td>
<td>0x804b983</td>
<td>2</td>
<td>0.00000</td>
</tr>
<tr>
<td>omp_set_lock</td>
<td>0x804bd94</td>
<td>0x804bdbb</td>
<td>0</td>
<td>0.01375</td>
</tr>
<tr>
<td></td>
<td>0x804bd94</td>
<td>0x804bdbb</td>
<td>1</td>
<td>0.013258</td>
</tr>
<tr>
<td></td>
<td>0x804bd94</td>
<td>0x804bdbb</td>
<td>2</td>
<td>0.012979</td>
</tr>
<tr>
<td></td>
<td>0x804bd94</td>
<td>0x804bdbb</td>
<td>3</td>
<td>0.010780</td>
</tr>
<tr>
<td>GOMP_barrier</td>
<td>0x804bdfb</td>
<td>0x804bdfb</td>
<td>3</td>
<td>0.018024</td>
</tr>
<tr>
<td>GOMP_barrier</td>
<td>0x804bdfb</td>
<td>0x804bdfb</td>
<td>2</td>
<td>0.010153</td>
</tr>
<tr>
<td>GOMP_barrier</td>
<td>0x804bdfb</td>
<td>0x804bdfb</td>
<td>1</td>
<td>0.010693</td>
</tr>
<tr>
<td>GOMP_barrier</td>
<td>0x804bdfb</td>
<td>0x804bdfb</td>
<td>0</td>
<td>0.008843</td>
</tr>
</tbody>
</table>
Performance?

> ./plain-ssca2.sh |& grep Time
Time taken for Scalable Data Gen. is 0.033507 sec.
Time taken for Kernel 1 is 0.001707 sec.
Time taken for Kernel 2 is 0.000193 sec.
Time taken for Kernel 3 is 0.000530 sec.
Time taken for Kernel 4 is 0.208041 sec.

> ./pgomp-aggregate.sh |& grep Time
Time taken for Scalable Data Gen. is 0.029894 sec.
Time taken for Kernel 1 is 0.003377 sec. (20x)
Time taken for Kernel 2 is 0.008760 sec. (45x)
Time taken for Kernel 3 is 0.010045 sec. (19x)
Time taken for Kernel 4 is 2.725435 sec. (13x)

Trace output is MUCH slower...
Location issues

Optimized code from SSCA2:

\ldots
8049186: call 80488c0 <GOMP_barrier@plt>
80491C4: jmp 80488c0 <GOMP_barrier@plt>
80491D0: call 80488c0 <GOMP_barrier@plt>
\ldots

Optimized code from our own test program:

804880E: call 8048660 <GOMP_critical_start@plt>
8048860: jmp 80485e0 <GOMP_critical_end@plt>
Conclusion

- PGOMP == easy instrumentation of Gnu-compiled OpenMP programs
- Initial prototype results are promising
- Much work still to do
  - Support OTF (Open Trace Format)
  - Support other tool's data formats (HPCToolkit)
  - Support POMP I/F? PAPI? Others?
  - Provide useful data processing scripts
    - At least some address->code mapping
"Any questions?"

prosportstickers.com