C++11 and three compilers: performance considerations

CERN openlab Summer Students Lightning Talks Sessions
Supervised by Pawel Szostek

Stephen Wang
Motivation

› “Surprisingly, C++11 feels like a new language” by Stroustrup.

› Improvements: Language usability, Mulithreading and other stuff.

› How about performance?

› Four questions: *Time measurement methods, For-loop efficiency, std::async, STL algorithms in parallel mode*

› GCC 4.9.0, ICC 15.0.0, Clang+LLVM 3.4

› -O2, -O3, -Ofast
Contribution

› Make four micro-benchmarks for each feature.
  Code: https://github.com/wangyichao/CPP11_Benchmarks

› **Automatize** the process, make results repeatable
  Python and bash script are used.
  Compile -> Run -> Table (3 compiler * 3 options * containers * algorithms …)

› **Use profiling tools** to check performance such as
  vectorization and threads.
  Perf, Intel Vtune, Likwid, even PMU

› Answer basic questions.
Conclusions

```cpp
auto start = std::chrono::steady_clock::now();
auto end = std::chrono::steady_clock::now();
int elapsed = std::chrono::duration_cast<std::chrono::nanoseconds>(end - start).count();
overhead.push_back(elapsed);
```
Conclusions

std::chrono (C++11) is a reliable time measurement.

```cpp
auto start = std::chrono::steady_clock::now();
 microseconds_sleep(index);
auto end = std::chrono::steady_clock::now();
int elapsed = std::chrono::duration_cast<std::chrono::nanoseconds>(end - start).count();
```
Conclusions

› Performance of for-loop varies with iteration method and container.

› Which containers are vectorized?

<table>
<thead>
<tr>
<th></th>
<th>array</th>
<th>std::array</th>
<th>std::list</th>
<th>std::set</th>
<th>std::vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>ICC</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
</tr>
<tr>
<td>Clang</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
</tr>
</tbody>
</table>

Table range-based for loop
std::async spawns threads when called and the computation is done immediately. (stdlibc++ from GCC)

double sum = 0;
auto handle1 = std::async(std::launch::async, fsum, v.begin(), v.begin()+distance);
Auto handle2 = std::async(...);
...  
...  
sum = handle1.get()+handle2.get()+handle3.get()+handle4.get();
Conclusions

GNU libstdc++ parallel speeds up part of STL algorithms, whose performance varies with containers.