Sweeten Your Yocto Build Times with Icecream

Joshua Watt
Sweeten Your Yocto Build Times with Icecream

Joshua Watt
About Me

- Worked for Garmin for the past 10 years
- Worked with Yocto for the past 4 years
  - I’ve been using icecream for most of that time
- Joshua.Watt@garmin.com
- JPEWhacker@gmail.com
Outline

- What is Icecream?
- Why use Icecream?
- How to use Icecream
- Maximizing Performance
- What’s next
- Conclusion
Outline

- What is Icecream?
- Why use Icecream?
- How to use Icecream
- Maximizing Performance
- What’s next
- Conclusion
What is Icecream?

- [https://github.com/icecc/icecream](https://github.com/icecc/icecream)
- A distributed compiler, similar to distcc
  - Unlike distcc, it uses single `scheduler` to dispatch jobs between the various nodes
Advantages of a central scheduler

- Quickly make node selections
- Better distribution of jobs across the cluster
- Holistic approach to scheduling tasks
- Easier cluster administration
What is Icecream?

Node A
- icecc (shim)
- iceccd (daemon)

Node B
- iceccd (daemon)

Node A Toolchain

Scheduler
Outline

- What is Icecream?
- **Why use Icecream?**
- How to use Icecream
- Maximizing Performance
- What’s next
- Conclusion
Performance Testing

- $ bitbake --runonly unpack core-image-minimal && bitbake core-image-minimal
- Results analyzed using buildstats tools
  - 15 test builds
- https://github.com/JPEWdev/oe-icecream-demo
Testing Environment

- Garmin’s icecream cluster:
  - ~21 compile nodes
  - ~184 total job capacity

- Test machine:
  - Quad core i7 CPU @ 3.4 GHz
  - 16 GB Memory
  - 1 TB spindle HD
  - Fedora 30
Results (CPU time)

<table>
<thead>
<tr>
<th>PKG</th>
<th>TASK</th>
<th>ABSDIFF</th>
<th>RELDIFF</th>
<th>CPUTIME1 -&gt; CPUTIME2</th>
</tr>
</thead>
<tbody>
<tr>
<td>qemu-system-native</td>
<td>do_compile</td>
<td>-1597.2s</td>
<td>-76.3%</td>
<td>2093.6s -&gt; 496.4s</td>
</tr>
<tr>
<td>gcc-cross-x86_64</td>
<td>do_compile</td>
<td>-1397.8s</td>
<td>-79.3%</td>
<td>1763.1s -&gt; 365.3s</td>
</tr>
<tr>
<td>linux-yocto</td>
<td>do_compile</td>
<td>-1269.8s</td>
<td>-67.0%</td>
<td>1895.2s -&gt; 625.4s</td>
</tr>
<tr>
<td>cmake-native</td>
<td>do_compile</td>
<td>-1001.4s</td>
<td>-90.9%</td>
<td>1101.2s -&gt; 99.8s</td>
</tr>
<tr>
<td>linux-yocto</td>
<td>do_compile_kernelmodules</td>
<td>-954.9s</td>
<td>-65.3%</td>
<td>1462.8s -&gt; 507.9s</td>
</tr>
<tr>
<td>qemu-native</td>
<td>do_compile</td>
<td>-817.0s</td>
<td>-81.7%</td>
<td>1000.3s -&gt; 183.3s</td>
</tr>
<tr>
<td>binutils-cross-x86_64</td>
<td>do_compile</td>
<td>-493.7s</td>
<td>-80.6%</td>
<td>612.6s -&gt; 118.9s</td>
</tr>
<tr>
<td>binutils-native</td>
<td>do_compile</td>
<td>-449.5s</td>
<td>-56.4%</td>
<td>796.8s -&gt; 347.4s</td>
</tr>
<tr>
<td>cmake-native</td>
<td>do_configure</td>
<td>-416.0s</td>
<td>-84.9%</td>
<td>490.1s -&gt; 74.2s</td>
</tr>
<tr>
<td>gcc-runtime</td>
<td>do_compile</td>
<td>-305.3s</td>
<td>-79.2%</td>
<td>385.6s -&gt; 80.3s</td>
</tr>
<tr>
<td>perl</td>
<td>do_compile</td>
<td>-222.1s</td>
<td>-40.3%</td>
<td>550.7s -&gt; 328.6s</td>
</tr>
<tr>
<td>libdnf-native</td>
<td>do_compile</td>
<td>-215.8s</td>
<td>-84.9%</td>
<td>254.0s -&gt; 38.2s</td>
</tr>
<tr>
<td>python3</td>
<td>do_compile</td>
<td>-203.2s</td>
<td>-84.7%</td>
<td>239.9s -&gt; 36.7s</td>
</tr>
<tr>
<td>openssl</td>
<td>do_compile</td>
<td>-194.2s</td>
<td>-58.6%</td>
<td>331.1s -&gt; 136.9s</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>elfutils-native</td>
<td>do_configure</td>
<td>40.5s</td>
<td>+357.7%</td>
<td>11.3s -&gt; 51.9s</td>
</tr>
<tr>
<td>pkgconfig-native</td>
<td>do_configure</td>
<td>45.4s</td>
<td>+95.5%</td>
<td>47.5s -&gt; 92.9s</td>
</tr>
<tr>
<td>openssl1-native</td>
<td>do_install</td>
<td>55.9s</td>
<td>+98.7%</td>
<td>56.6s -&gt; 112.5s</td>
</tr>
<tr>
<td>glibc-locale</td>
<td>do_package</td>
<td>160.5s</td>
<td>+13.7%</td>
<td>1175.2s -&gt; 1335.7s</td>
</tr>
<tr>
<td>gmp-native</td>
<td>do_compile</td>
<td>275.5s</td>
<td>+339.6%</td>
<td>81.1s -&gt; 356.7s</td>
</tr>
</tbody>
</table>

Cumulative cputime:
-12419.0s  -40.8%  8:26:48.2 (30408.2s) -> 4:59:49.2 (17989.2s)
Average cputime (n = 15)

* p < 0.05

- `do_compile`
- `do_compile_kemelmodes`
- `do_configure`
- `do_install`
- `other`
- `overall`

- Without Icecream
- With Icecream
## Results (Wall Time)

### Without vs. With

<table>
<thead>
<tr>
<th>PKG</th>
<th>TASK</th>
<th>ABSDIFF</th>
<th>RELDIFF</th>
<th>WALLTIME1 -&gt; WALLTIME2</th>
</tr>
</thead>
<tbody>
<tr>
<td>qemu-system-native</td>
<td>do_compile</td>
<td>-522.9s</td>
<td>-70.0%</td>
<td>747.4s -&gt; 224.5s</td>
</tr>
<tr>
<td>linux-yocto</td>
<td>do_compile_kernelmodules</td>
<td>-430.8s</td>
<td>-76.4%</td>
<td>563.7s -&gt; 132.8s</td>
</tr>
<tr>
<td>perl</td>
<td>do_install ptest base</td>
<td>-320.1s</td>
<td>-71.9%</td>
<td>445.5s -&gt; 125.4s</td>
</tr>
<tr>
<td>glibc-locale</td>
<td>do_package</td>
<td>-302.1s</td>
<td>-51.4%</td>
<td>622.4s -&gt; 302.3s</td>
</tr>
<tr>
<td>gcc-cross-x86_64</td>
<td>do_compile</td>
<td>-315.8s</td>
<td>-64.3%</td>
<td>490.9s -&gt; 175.1s</td>
</tr>
<tr>
<td>qemu-native</td>
<td>do_compile</td>
<td>-281.3s</td>
<td>-83.7%</td>
<td>336.1s -&gt; 54.8s</td>
</tr>
<tr>
<td>linux-yocto</td>
<td>do_kernel_configcheck</td>
<td>-277.0s</td>
<td>-79.8%</td>
<td>347.1s -&gt; 70.1s</td>
</tr>
<tr>
<td>cmake-native</td>
<td>do_compile</td>
<td>-262.0s</td>
<td>-70.1%</td>
<td>373.7s -&gt; 111.8s</td>
</tr>
<tr>
<td>gcc-runtime</td>
<td>do_configure</td>
<td>-201.8s</td>
<td>-50.9%</td>
<td>396.4s -&gt; 194.6s</td>
</tr>
<tr>
<td>libxml2</td>
<td>do_package</td>
<td>-201.1s</td>
<td>-82.0%</td>
<td>245.1s -&gt; 44.1s</td>
</tr>
<tr>
<td>libxcb</td>
<td>do_package_write_rpm</td>
<td>-176.3s</td>
<td>-62.5%</td>
<td>282.1s -&gt; 105.8s</td>
</tr>
<tr>
<td>gettext-native</td>
<td>do_configure</td>
<td>-175.0s</td>
<td>-43.3%</td>
<td>404.1s -&gt; 229.1s</td>
</tr>
<tr>
<td>gcc-runtime</td>
<td>do_compile</td>
<td>-172.4s</td>
<td>-80.1%</td>
<td>215.3s -&gt; 42.9s</td>
</tr>
<tr>
<td>nss-native</td>
<td>do_compile</td>
<td>-162.6s</td>
<td>-50.4%</td>
<td>322.5s -&gt; 159.8s</td>
</tr>
<tr>
<td>elfutils-native</td>
<td>do_configure</td>
<td>89.6s</td>
<td>+267.4%</td>
<td>33.5s -&gt; 123.1s</td>
</tr>
<tr>
<td>perl</td>
<td>do_package</td>
<td>123.7s</td>
<td>+44.6%</td>
<td>277.2s -&gt; 401.0s</td>
</tr>
<tr>
<td>shared-mime-info-native</td>
<td>do_install</td>
<td>135.0s</td>
<td>+238.0%</td>
<td>56.7s -&gt; 191.8s</td>
</tr>
<tr>
<td>binutils-native</td>
<td>do_install</td>
<td>171.9s</td>
<td>+172.0%</td>
<td>99.9s -&gt; 271.7s</td>
</tr>
<tr>
<td>glibc</td>
<td>do_install</td>
<td>206.9s</td>
<td>+64.8%</td>
<td>319.2s -&gt; 526.2s</td>
</tr>
</tbody>
</table>

**Cumulative walltime:**

-9335.0s  -29.6%  8:46:07.8 (31567.8s) -> 6:10:32.8 (22232.8s)
Average walltime (n = 15)

* p < 0.05

* Without Icecream
* With Icecream
Average Elapsed Build Time (n = 15)

- Average without Icecream: 5433.9 s
- Average with Icecream: 4369.1 s

Percent Change: -20%
Difference: -22%

Average CPU Usage (n = 15)

- Average without Icecream: 59.3%
- Average with Icecream: 37.3%

p < 0.05
## Results (Elapsed Time)

<table>
<thead>
<tr>
<th></th>
<th>ABSDIFF</th>
<th>RELDIFF</th>
<th>ELAPSED1 -&gt; ELAPSED2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed time:</td>
<td>-1185.47s</td>
<td>-24.76%</td>
<td>4787.41s -&gt; 3601.94s</td>
</tr>
<tr>
<td>CPU usage:</td>
<td>-19.7%</td>
<td>-29.40%</td>
<td>67.0% -&gt; 47.3%</td>
</tr>
</tbody>
</table>
## Results (Per-task totals)

<table>
<thead>
<tr>
<th>Task</th>
<th>Cumulative cputime</th>
<th>Cumulative walltime</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>do_configure</strong></td>
<td>-341.1s -9.8%</td>
<td>57:51.9 (3471.9s) -&gt; 52:10.7 (3130.7s)</td>
</tr>
<tr>
<td></td>
<td>-1903.0s -21.8%</td>
<td>2:25:48.8 (8748.8s) -&gt; 1:54:05.8 (6845.8s)</td>
</tr>
<tr>
<td><strong>do_compile</strong></td>
<td>-11284.6s -60.1%</td>
<td>5:13:08.1 (18788.1s) -&gt; 2:05:03.5 (7503.5s)</td>
</tr>
<tr>
<td></td>
<td>-4202.6s -47.0%</td>
<td>2:29:08.5 (8948.5s) -&gt; 1:19:05.8 (4745.8s)</td>
</tr>
<tr>
<td><strong>do_install</strong></td>
<td>13.2s +1.4%</td>
<td>16:00.5 (960.5s) -&gt; 16:13.7 (973.7s)</td>
</tr>
<tr>
<td></td>
<td>187.7s +8.3%</td>
<td>37:47.4 (2267.4s) -&gt; 40:55.1 (2455.1s)</td>
</tr>
<tr>
<td><strong>do_package_write_rpm</strong></td>
<td>8.4s +0.5%</td>
<td>28:54.3 (1734.3s) -&gt; 29:02.7 (1742.7s)</td>
</tr>
<tr>
<td></td>
<td>-600.3s -18.4%</td>
<td>54:14.8 (3254.8s) -&gt; 44:14.5 (2654.5s)</td>
</tr>
</tbody>
</table>
Why use Icecream?

- Total elapsed build time reduction of 20%
- Computer is more responsive when building due to lower overall CPU usage
- Full rebuilds of individual recipes can be much faster
Results Analysis

- Total build time performance improvement is marginal
  - Better system responsiveness when building
- Full rebuilds of individual recipes can be much faster
Outline

- What is Icecream?
- Why use Icecream?
- **How to use Icecream**
- Maximizing Performance
- What’s next
- Conclusion
How to enable Icecream?

Enabling Icecream is as easy(*) as adding the following to local.conf:

```
INHERIT += "icecc"
ICECC_PARALLEL_MAKE = "-j 24"
```
SDK Support

- Icecream also integrates with your traditional SDK
  - Building an SDK with `INHERIT += "icecc"` will automatically include support for Icecream
  - Icecream will be enabled for the SDK if the host has `icecc` when the SDK is installed

```
$ ./poky-glibc-x86_64-core-image-minimal-core2-64-qemux86-64-toolchain-2.7+snapshot.sh
Poky (Yocto Project Reference Distro) SDK installer version 2.7+snapshot
========================================================================
Enter target directory for SDK (default: /opt/poky/2.7+snapshot):
You are about to install the SDK to "/opt/poky/2.7+snapshot". Proceed [Y/n]? y
Extracting SDK..........................done
Setting it up...done
Setting up IceCream distributed compiling...
creating /opt/.../poky-glibc-x86_64-x86_64-poky-linux-2.7+snapshot-20190722.tar.gz
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment
setup script e.g.
$ . /opt/poky/2.7+snapshot/environment-setup-core2-64-poky-linux
```
Combining with sstate

- Get sstate working first
  - Icecream will *never* give as much benefit as a well populated sstate server

**Problem:** Icecream and sstate can be combined, however inheriting `icecc.bbclass` changes most taskhashes

**Solution:** Always inherit `icecc.bbclass` and use `ICECC_DISABLED ?= "1"` to turn off icecream
Outline

- What is Icecream?
- Why use Icecream?
- How to use Icecream
- **Maximizing Performance**
- What’s next
- Conclusion
Blacklisting

- Some massaging of `ICECC_USER_PACKAGE_BL` is necessary
  - There needs to be a better way to do this
Network Performance

- The test cluster has gigabit ethernet between all nodes
- Using 100 Mbps or less network speed is not likely to give good results
- Wi-Fi probably won’t work well either
Keep Up to Date with Upstream

- New versions of GCC generally requires updates to Icecream for bug fixes
- Upstream only releases about once a year
- Most of the changes are in the client side icecc shim and scheduler; the daemon is less important
- We build in a Docker container with a patched icecc shim
Use a dedicated scheduler

- The scheduler doesn’t take much CPU, but it is sensitive to latency
- Eliminates scheduler ping pong
Avoid Virtual Machines

- Virtual Machines as compile nodes seem to be particularly bad for performance
  - Blacklist or mark as “No Remote”
Remote Preprocessing

- Icecream has the option of preprocessing remotely (ICECC_REMOTE_CPP), which improves performance even more
  - GCC has lots of bugs related to “-fdirectives-only”, which currently makes this impractical in the general case.
    - [https://gcc.gnu.org/bugzilla/buglist.cgi?quicksearch=directives-only](https://gcc.gnu.org/bugzilla/buglist.cgi?quicksearch=directives-only)
Outline

- What is Icecream?
- Why use Icecream?
- How to use Icecream
- Maximizing Performance
- What’s next
- Conclusion
What’s Next?

- Build Icecream client shim in OE-core (e.g. icecc-native)?
- Clang support?
- ccache support?
- Gather more data from other clusters
- Fix up GCC’s `-fdirectives-only` support
- eSDK support
Outline

● What is Icecream?
● Why use Icecream?
● How to use Icecream
● Maximizing Performance
● What’s next
● Conclusion
Conclusion

- Icecream distributes jobs compiles with a centralized scheduler
- We saw a 20% build time improvement with Icecream
- There are many ways to get involved if you would like to improve the experience
Useful Links

- https://www.openembedded.org/wiki/Using_IceCC
- https://github.com/icecc/icecream
- Icecream Monitors:
  - https://github.com/icecc/icemon
  - https://github.com/JPEWdev/icecream-sundae (shameless plug)
- Replicate my test:
  - https://github.com/JPEWdev/oe-icecream-demo
Thanks

- Garmin
- Icecream Developers
- Open Emebedded Developers
- Kristin Watt, PhD
Questions?