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

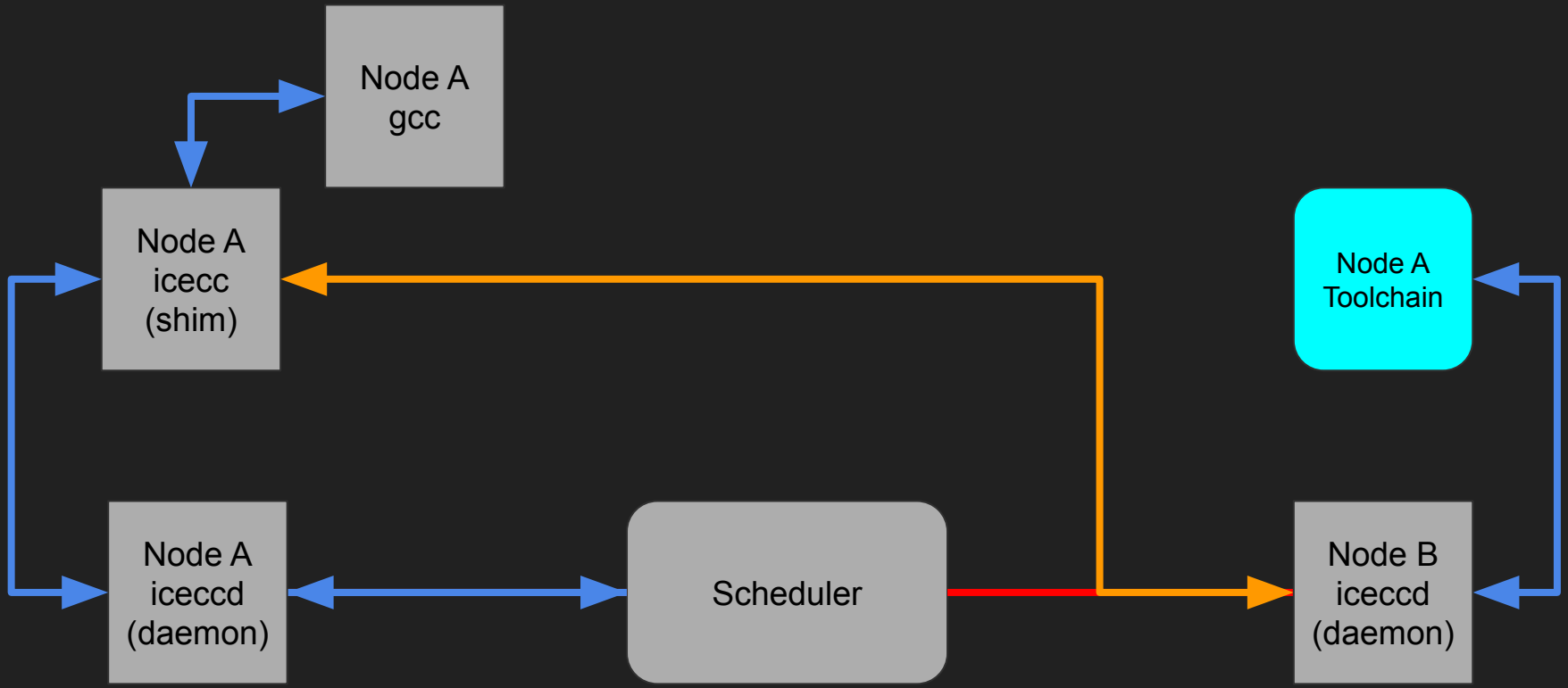
What is Icecream?

- https://github.com/icecc/icecream
- 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?



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

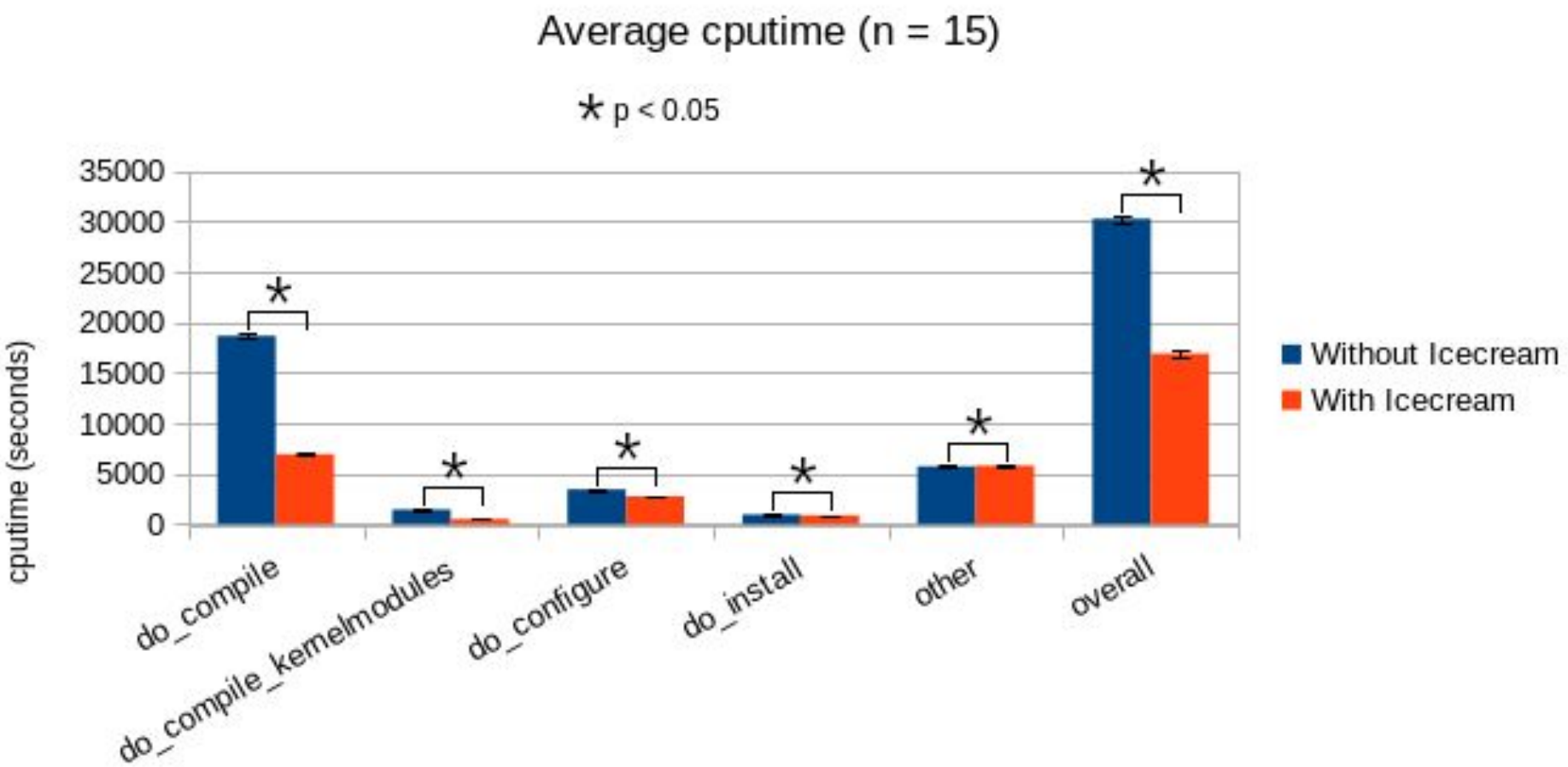
- Garmin’s icecream cluster:
  - ~21 compile nodes
  - ~184 total job capacity
- Test machine:
  - Quad core i7 CPU @ 3.4 GHz
  - 16 GB Memory
  - 1 TB spindle HD
  - Fedora 30

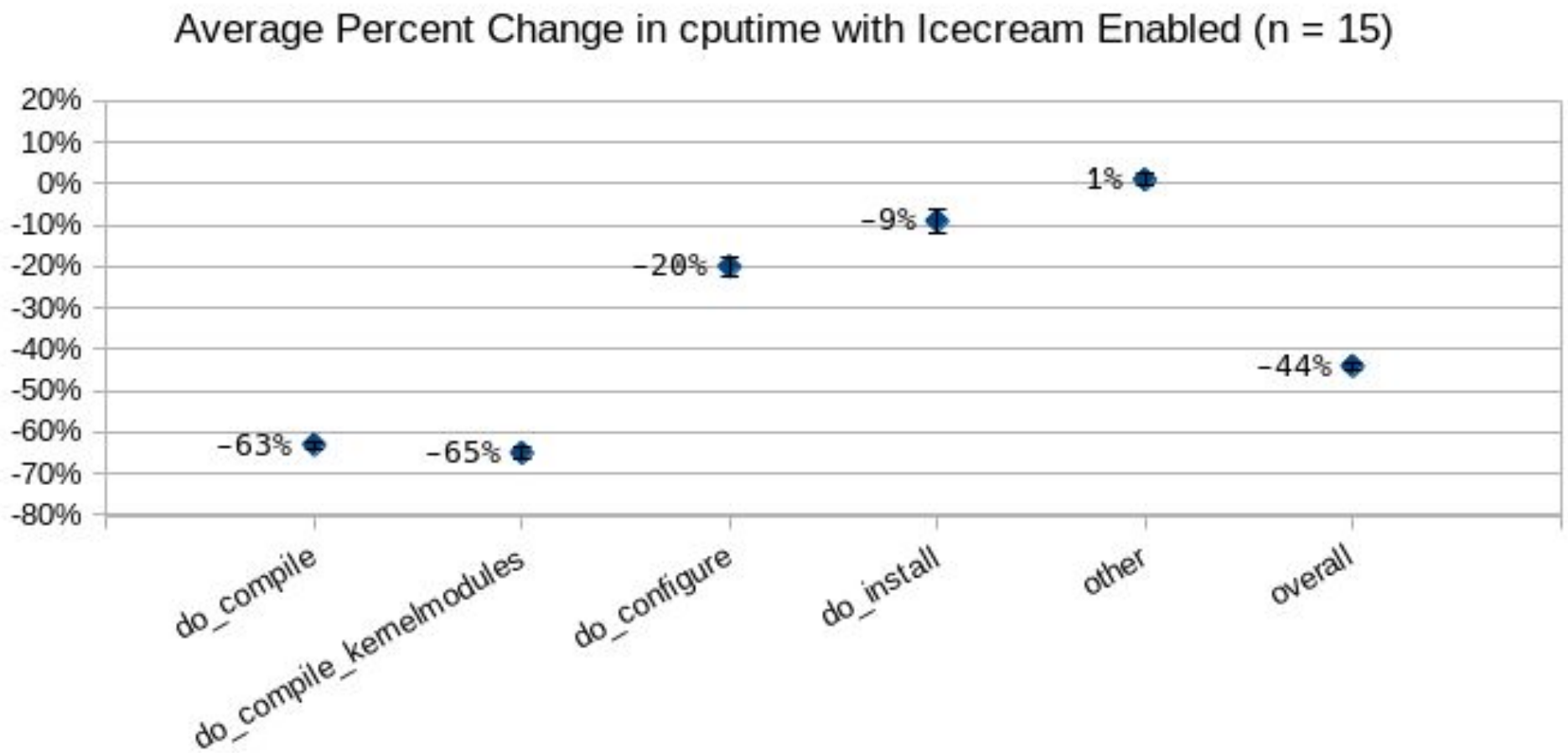
Results (CPU time)

Without Icecream -> With Icecream

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

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



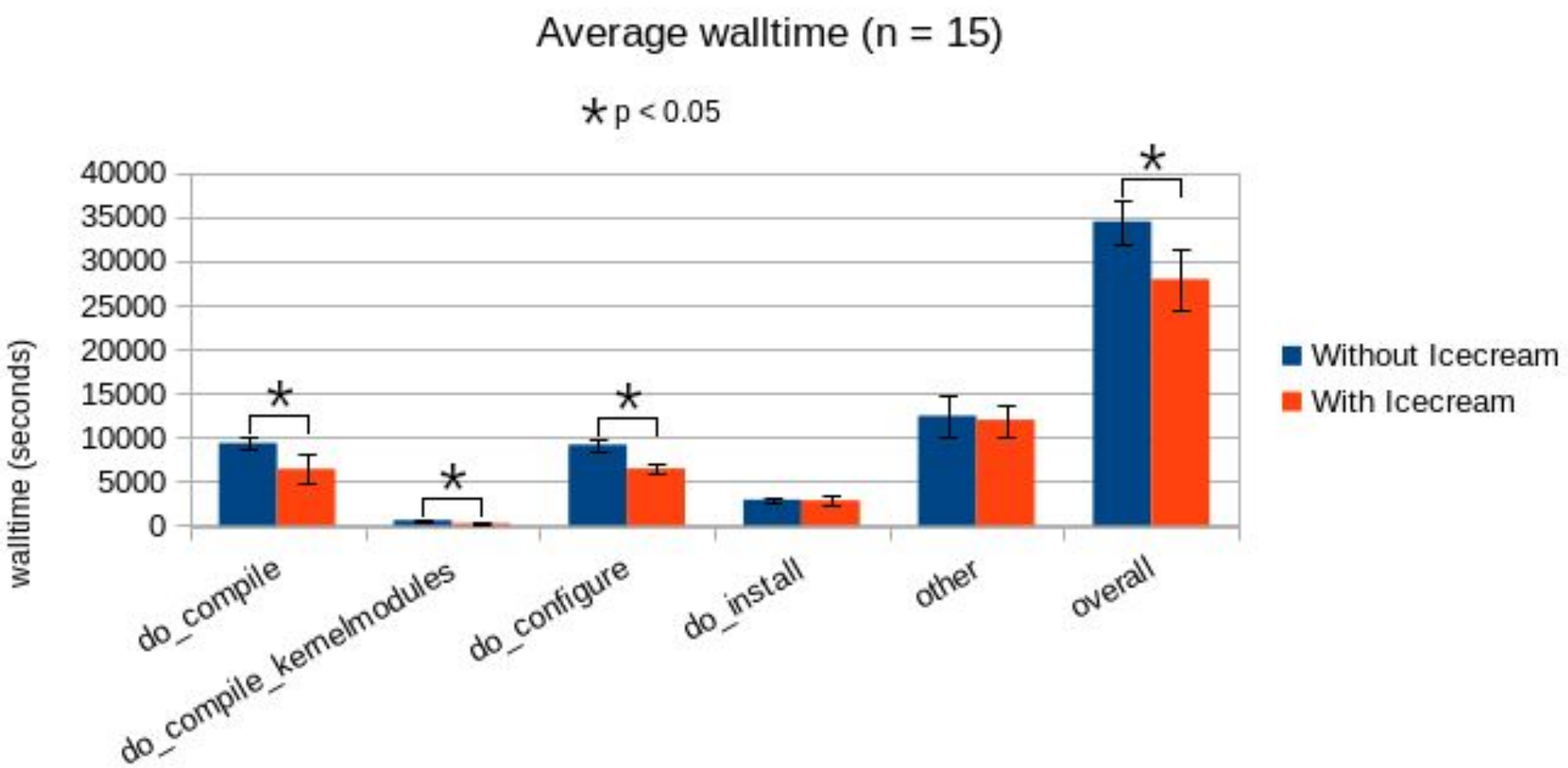


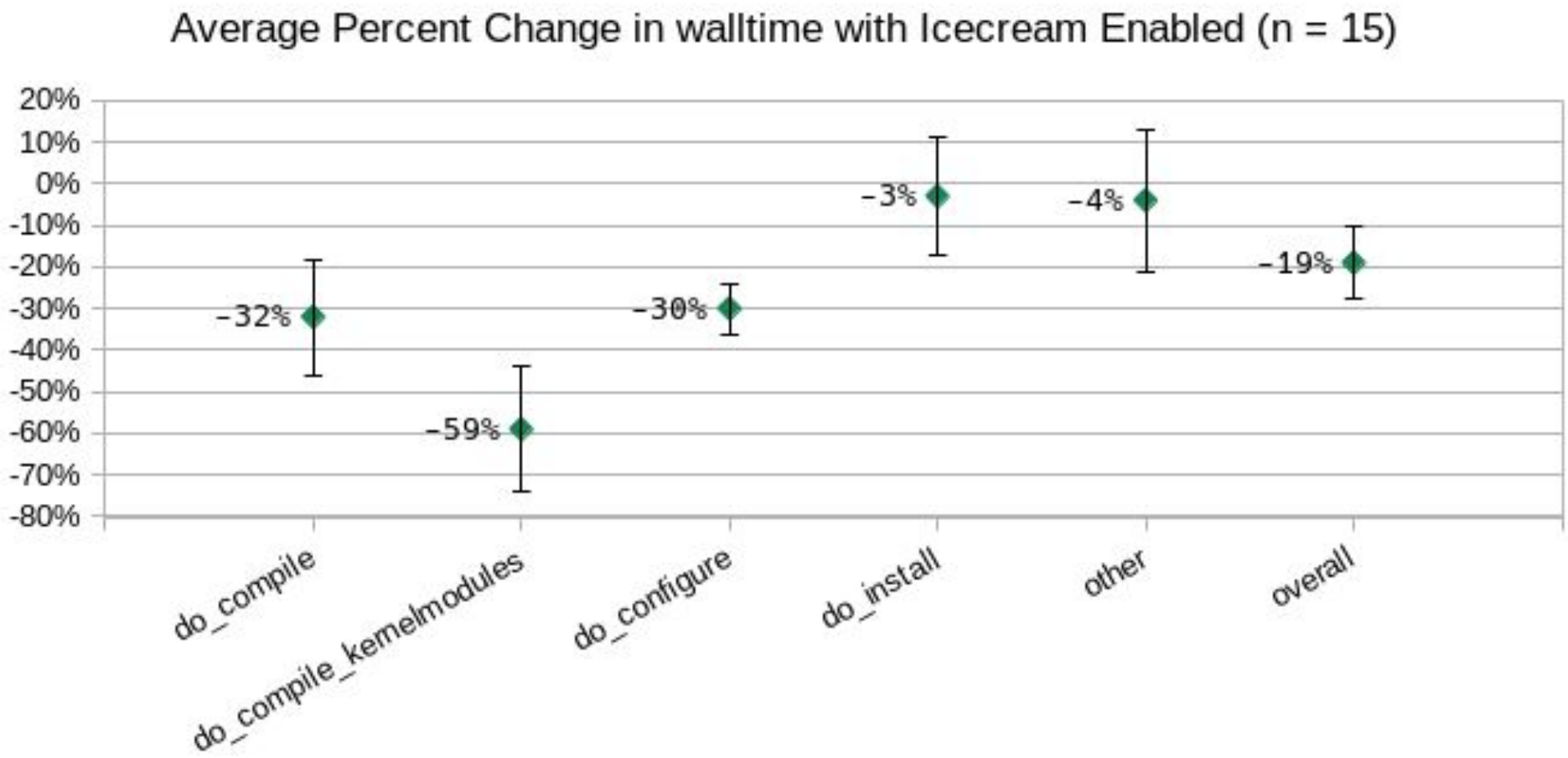
Results (Wall Time)

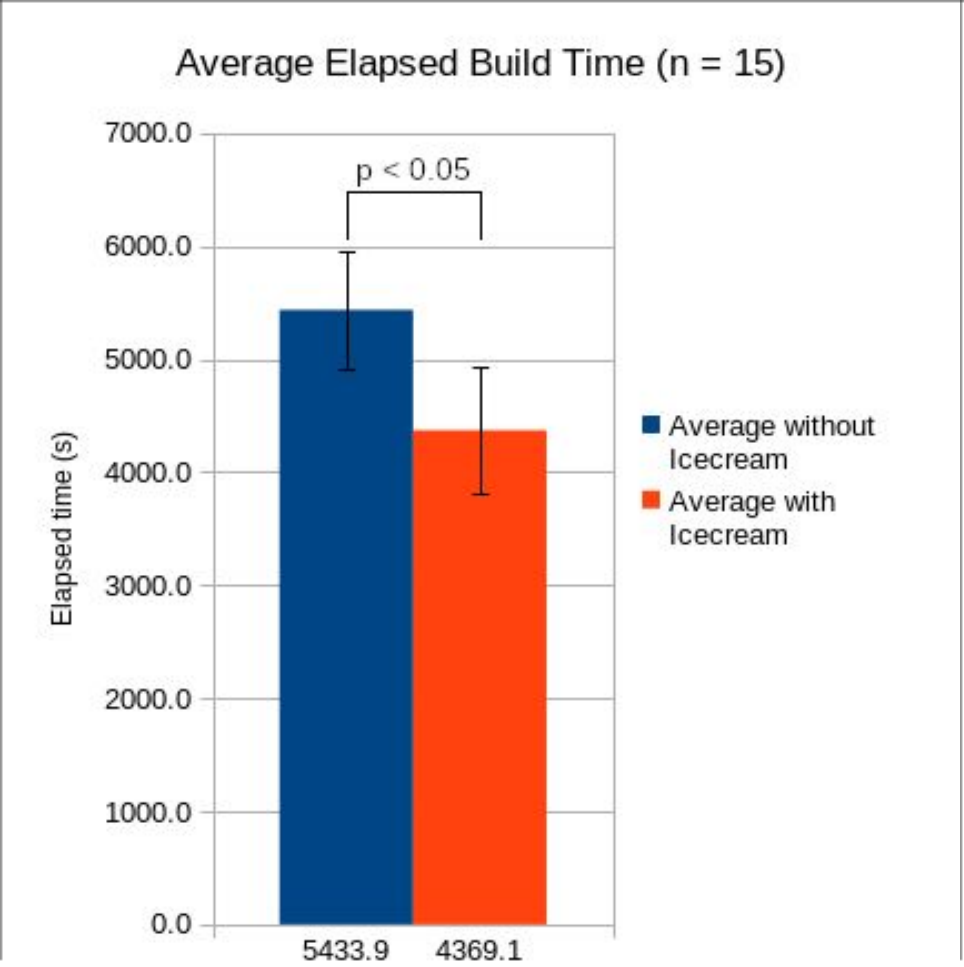
Without Icecream -> With Icecream

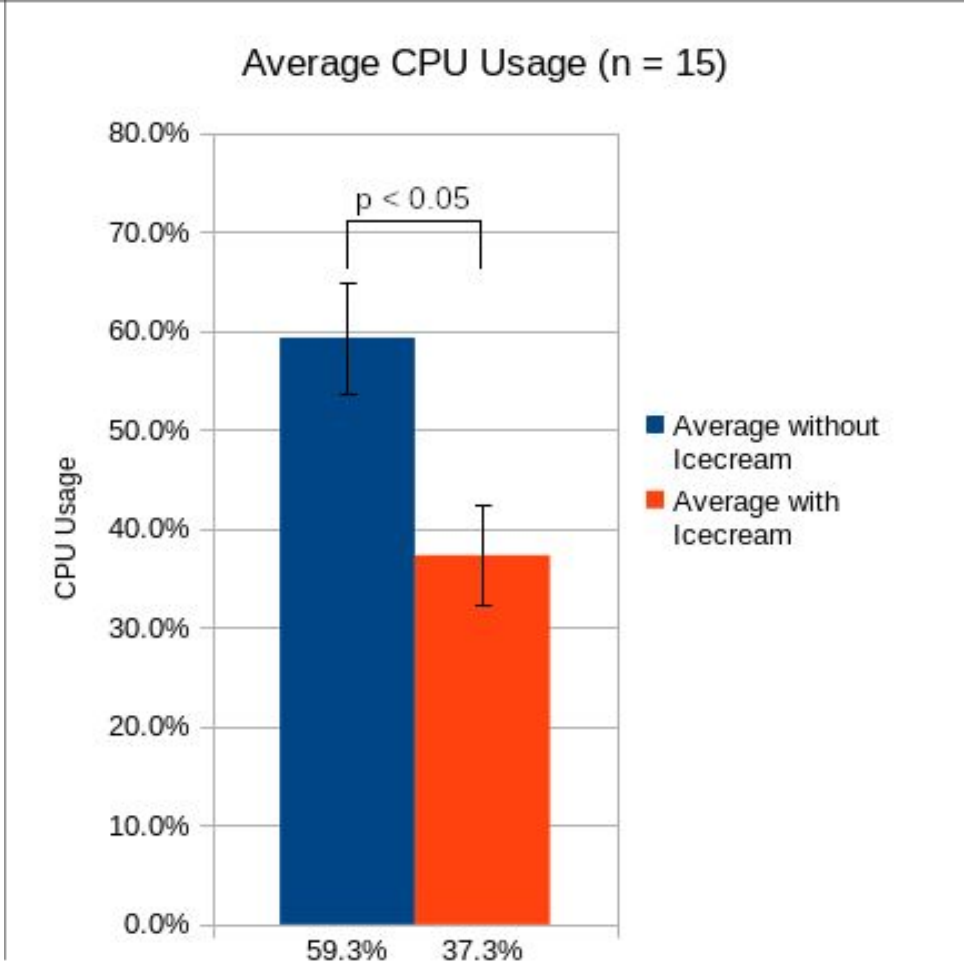
PKG	TASK	ABSDIFF	RELDIFF	WALLTIME1 -> WALLTIME2
qemu-system-native	do_compile	-522.9s	-70.0%	747.4s -> 224.5s
linux-yocto	do_compile_kernelmodules	-430.8s	-76.4%	563.7s -> 132.8s
perl	do_install_ptest_base	-320.1s	-71.9%	445.5s -> 125.4s
glibc-locale	do_package	-320.1s	-51.4%	622.4s -> 302.3s
gcc-cross-x86_64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

Cumulative walltime:
-9335.0s -29.6% 8:46:07.8 (31567.8s) -> 6:10:32.8 (22232.8s)









Percent Change: -20%

Difference: -22%

Results (Elapsed Time)

```
                ABSDIFF   RELDIFF  ELAPSED1 -> ELAPSED2
Elapsed time: -1185.47s   -24.76%  4787.41s -> 3601.94s
CPU usage:       -19.7%   -29.40%     67.0% -> 47.3%
```

Results (Per-task totals)

```
do_configure:
Cumulative cputime:
    -341.1s     -9.8%       57:51.9 (3471.9s) -> 52:10.7 (3130.7s)
Cumulative walltime:
   -1903.0s    -21.8%     2:25:48.8 (8748.8s) -> 1:54:05.8 (6845.8s)

do_compile:
Cumulative cputime:
  -11284.6s    -60.1%    5:13:08.1 (18788.1s) -> 2:05:03.5 (7503.5s)
Cumulative walltime:
   -4202.6s    -47.0%     2:29:08.5 (8948.5s) -> 1:19:05.8 (4745.8s)

do_install:
Cumulative cputime:
      13.2s     +1.4%        16:00.5 (960.5s) -> 16:13.7 (973.7s)
Cumulative walltime:
     187.7s     +8.3%       37:47.4 (2267.4s) -> 40:55.1 (2455.1s)

do_package_write_rpm:
Cumulative cputime:
       8.4s     +0.5%       28:54.3 (1734.3s) -> 29:02.7 (1742.7s)
Cumulative walltime:
    -600.3s    -18.4%       54:14.8 (3254.8s) -> 44:14.5 (2654.5s)
```

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

Outline

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

What's Next?

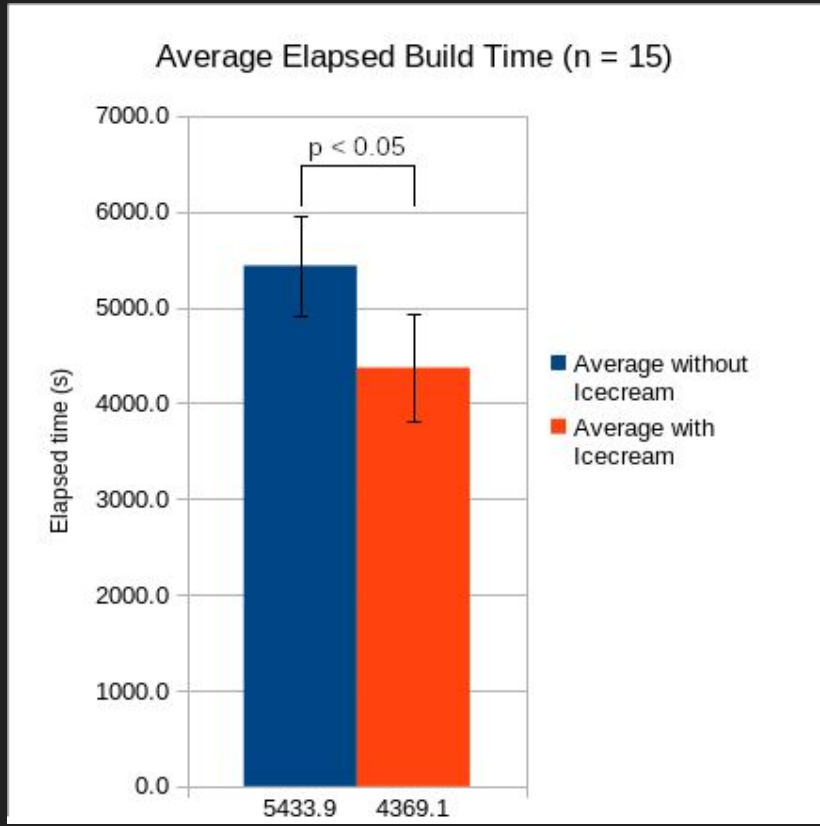
- Build Icecream client shim in OE-core (e.g. icecc-native)?
- Clang support?
- ccache support?
- Gather more data from other clusters
- Fix up GCC's `-fdirectives-only` support
- eSDK support

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 expirence



Percent Change: -20%

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?