Supporting architecture psABIs with GNU Guix

Efraim Flashner
This program interpreter self-identifies as:
/gnu/store/...-glibc-2.35/lib/ld-linux-x86-64.so.2

Shared library search path:
(libraries located via /etc/ld.so.cache)
/gnu/store/...-glibc-2.35/lib (system search path)

Subdirectories of glibc-hwcaps directories, in
priority order:
  x86-64-v4
  x86-64-v3 (supported, searched)
  x86-64-v2 (supported, searched)
Where are the directories?
Where are the directories?

glibc.git $ git grep glibc-hwcaps
elf/tst-glibc-hwcaps-2-cache.script:
mkdirp 0770 $L/glibc-hwcaps/x86-64-v2
mkdirp 0770 $L/glibc-hwcaps/x86-64-v3
cp $B/elf/libx86-64-isa-level-3.so \
   $L/glibc-hwcaps/x86-64-v2/libx86-64-isa-level.so
cp $B/elf/libx86-64-isa-level-4.so \
   $L/glibc-hwcaps/x86-64-v3/libx86-64-isa-level.so
Where are the directories?

glibc.git $ git grep const\char\_dl_hwcaps_subdirs
sysdeps/powerpc/powerpc64/le/dl-hwcaps-subdirs.c: \n  const char _dl_hwcaps_subdirs[] = \n  "power10:power9";
sysdeps/s390/s390-64/dl-hwcaps-subdirs.c: \n  const char _dl_hwcaps_subdirs[] = \n  "z16:z15:z14:z13";
sysdeps/x86_64/dl-hwcaps-subdirs.c: \n  const char _dl_hwcaps_subdirs[] = \n  "x86-64-v4:x86-64-v3:x86-64-v2";
Where are the directories?

- /lib/
- /lib/glibc-hwcaps/x86-64-v2/
- /lib/glibc-hwcaps/x86-64-v3/
- /lib/glibc-hwcaps/x86-64-v4/
Where are the directories?

- /lib/
- /lib/glibc-hwcaps/power9/
- /lib/glibc-hwcaps/power10/

* little-endian only
Where are the directories?

- /lib/
- /lib/glibc-hwcaps/z13/
- /lib/glibc-hwcaps/z14/
- /lib/glibc-hwcaps/z15/
- /lib/glibc-hwcaps/z16/
Directed Acyclic Graph
Is it worth it?
Is it worth it?

Does it matter?
ncdu x86-64-v1 vs x86-64-v3

- 238  »  mov····%rcx,0x20(%rax)
- 239  »  movups·-0x68(%rbp),%xmm0
- 240  »  movups·-0x58(%rbp),%xmm1
- 241  »  movups·%xmm1,0x10(%rax)
- 242  »  movups·%xmm0,(%rax)
- 243  »  mov····%fs:0x28,%rax
- 244  »  mov····-0x8(%rbp),%rcx
- 245  »  cmp····%rcx,%rax
- 246  »  jne····2391e1·<sinf@@Base-0xeea5f>
- 247  »  mov····-0xa0(%rbp),%rax
- 248  »  add····$0xb0,%rsp
- 249  »  pop····%rbp

- 250  »  retq

- 251  »  call···332820·<__stack_chk_fail@plt>
- 252  »  cs·nopw·0x0(%rax,%rax,1)

- 238  »  mov····%rcx,0x20(%rax)
- 239  »  vmovups·-0x68(%rbp),%ymm0
- 240  »  vmovups·-0x58(%rbp),%ymm1
- 241  »  vmovups·%ymm1,0x10(%rax)
- 242  »  vmovups·%ymm0,(%rax)
- 243  »  mov····%fs:0x28,%rax
- 244  »  mov····-0x8(%rbp),%rcx
- 245  »  cmp····%rcx,%rax
- 246  »  jne····23927e·<sinf@@Base-0xef712>
- 247  »  mov····-0xa0(%rbp),%rax
- 248  »  add····$0xb0,%rsp
- 249  »  pop····%rbp
- 250  »  vzeroupper·
- 251  »  call···333c40·<__stack_chk_fail@plt>
- 252  »  cs·nopw·0x0(%rax,%rax,1)
Let’s see some code
(define (gsl-hwabi psabi)
  (package/inherit gsl
    (name (string-append "gsl-" psabi))
    (arguments
      (substitute-keyword-arguments (package-arguments gsl)
        (;;; make-flags
          (append (list (string-append "CFLAGS=-march=" #$psabi)
              (string-append "CXXFLAGS=-march=" #$psabi))
            #$flags))
        (;;; configure-flags
          (append (list (string-append "--libdir=" #$output
              "/lib/glibc-hwcaps/" #$psabi))
            #$flags))
    ;; The building machine can't necessarily run the code produced.
    (;;; tests
      (_ #t) #f)
    (;;; phases
      (add-after 'install 'remove-extra-files
        (lambda _
          (for-each (lambda (dir)
              (delete-file-recursively (string-append #$output dir)))
            (list (string-append "/lib/glibc-hwcaps/" #$psabi "/pkgconfig")
              "/bin" "/include" "/share"))))))))
(supported-systems (list "x86_64-linux" "powerpc64le-linux")
(properties ((hidden? . #t)
      (tunable? . #f))))
This copy of gsl will automatically use the libraries that target the x86_64 psABI which the hardware supports.

(define-public gsl-hwcaps
  (package/inherit gsl
    (name "gsl-hwcaps")
    (arguments
      (substitute-keyword-arguments (package-arguments gsl)
        ((#:phases phases #~%standard-phases)
          #~(modify-phases #$phases
            (add-after 'install 'install-optimized-libraries
              (lambda* (#:key inputs outputs #:allow-other-keys)
                (let ((hwcaps "/lib/glibc-hwcaps/"))
                  (for-each
                    (lambda (psabi)
                      (copy-recursively
                        (string-append (assoc-ref inputs (string-append "gsl-" psabi))
                          hwcaps psabi)
                        (string-append #$output hwcaps psabi)))
                  (list "x86-64-v2" "x86-64-v3" "x86-64-v4"))))))))
    (native-inputs
      (modify-inputs (package-native-inputs gsl)
        (append (gsl-hwabi "x86-64-v2")
                (gsl-hwabi "x86-64-v3")
                (gsl-hwabi "x86-64-v4")))
      (supported-systems (list "x86_64-linux")
        (properties `((tunable? . #f))))))
$ tree /gnu/store/j1w1sdd6f3jz61mv5hcrf98imfn9bglsls-gsl-hwcaps-2.7.1/
  bin
   ├── gsl-config
   │    └── gsl-histogram
   │         └── gsl-randist
   └── etc
       └── ld.so.cache
  include
   └── gsl
  lib
  share
   ├── aclocal
   │    └── gsl.m4
   └── doc
      └── gsl-hwcaps-2.7.1
          └── COPYING
      └── info
          └── gsl-ref.info.gz
  man
$ tree /gnu/store/j1w1sdd6f3jz61mv5hcrf98imfn9bgls-gsl-hwcaps-2.7.1/lib/
  glibc-hwcaps
    ├── x86-64-v2
    │    ├── libgsl.la
    │    ├── libgsl.so -> libgsl.so.27.0.0
    │    ├── libgsl.so.27 -> libgsl.so.27.0.0
    │    └── libgsl.so.27.0.0
    ├── x86-64-v3
    │    ├── libgsl.la
    │    ├── libgsl.so -> libgsl.so.27.0.0
    │    ├── libgsl.so.27 -> libgsl.so.27.0.0
    │    └── libgsl.so.27.0.0
    └── x86-64-v4
        ├── libgsl.la
        ├── libgsl.so -> libgsl.so.27.0.0
        ├── libgsl.so.27 -> libgsl.so.27.0.0
        └── libgsl.so.27.0.0
  └── pkgconfig
      └── gsl.pc
Using the custom packages

(define use-glibc-hwcaps
 (package-input-rewriting/spec
 ;; Replace some packages with ones built targeting custom packages build
 ;; with glibc-hwcaps support.
 ("gsl" . ,(const gsl-hwcaps))
 ("sdsl-lite" . ,(const sdsl-lite-hwcaps))
 ("seqwish" . ,(const seqwish-hwcaps))
 ("odgi" . ,(const odgi-hwcaps))
 ("wfmash" . ,(const wfmash-hwcaps))))

(define-public pggb-with-hwcaps
 (package
 (inherit (use-glibc-hwcaps pggb))
 (name "pggb-with-hwcaps")))
Is it worth it?

Only sometimes
Experiments in Performance

Distribution focused analysis

x86-64-v3: Mixed Bag of Performance

The pursuit of performance has long been sought after by advanced users, from custom compiling select packages to building your whole system from source. The inclusion of new variations to the default `x86_64` in the psABI has seen this extended to the distribution level, where some distributions are looking (or in the process) to include higher levels such as `x86-64-v2` or `x86-64-v3`. The belief in the performance improvements have even resulted in new distributions being created to maximize performance.
A Lot of Hype and Marketing

One of the newer entries into the performance arena is CachyOS, which rebuilds some of the Arch Linux repos with x86-64-v3 (plus some further modifications to performance sensitive packages). There are other modifications as well (which you can read on their website), but for this we are only interested in the impact of providing these packages without any of the other changes.

On the wiki it states that "if x86-64-v3 is detected it will automatically use the optimized packages, which yields more than 10% performance improvement." This is a very powerful statement...but is it true? In fairness, they are telling new users on discord that not all packages will benefit from using x86-64-v3, but adding caveats doesn't make it sound nearly as good. Do package optimizations bring better performance across the board?
Inspecting the CachyOS packages, it appears they are setting `-march=x86-64-v3 -mpclmul -O3` vs `-march=x86-64 -O2` (other flags being the upstream Arch Linux defaults) in Arch Linux, so we are testing wider optimizations than just switching to `x86-64-v3`. There are also some instances of `-mtune=skylake` which may have been set at one stage. Increasing the `-march` level and optimization levels are usually touted as the easiest ways to improve performance.
<table>
<thead>
<tr>
<th></th>
<th>Arch – Time</th>
<th>Power</th>
<th>CachyOS – Time</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compress Kernel (-3)</td>
<td>47.01</td>
<td>689.8</td>
<td>-2.5%</td>
<td>-1.5%</td>
</tr>
<tr>
<td>Compress Kernel (-9)</td>
<td>49.80</td>
<td>736.8</td>
<td>-1.5%</td>
<td>-1.4%</td>
</tr>
<tr>
<td>Decompress Kernel</td>
<td>15.31</td>
<td>225.5</td>
<td>7.1%</td>
<td>6.9%</td>
</tr>
<tr>
<td></td>
<td>Arch - Time</td>
<td>Power</td>
<td>CacheOS - Time</td>
<td>Power</td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
<td>-------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>Decode</td>
<td>0.71</td>
<td>10.5</td>
<td>-10.1%</td>
<td>-12.1%</td>
</tr>
<tr>
<td>Encode (-3)</td>
<td>0.90</td>
<td>13.3</td>
<td>-15.0%</td>
<td>-12.4%</td>
</tr>
<tr>
<td>Encode (-8)</td>
<td>2.95</td>
<td>49.1</td>
<td>-20.2%</td>
<td>-23.4%</td>
</tr>
</tbody>
</table>

To no surprise, flac benefits a lot from optimization. The source code already includes AVX2 runtime functions to improve the performance without requiring it to be enabled via the -march flag. Quite an improvement given it will already use AVX2 on the Arch build.
<table>
<thead>
<tr>
<th></th>
<th>Arch – Time</th>
<th>Power</th>
<th>CachyOS – Time</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gawk</td>
<td>2.88</td>
<td>42.9</td>
<td>-2.3%</td>
<td>-0.7%</td>
</tr>
</tbody>
</table>

Not a commonly thought about package for benchmarking and only a slim improvement.
<table>
<thead>
<tr>
<th>gzip</th>
<th>Arch – Time</th>
<th>Power</th>
<th>CachyOS – Time</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compress Kernel (-3)</td>
<td>10.33</td>
<td>152.1</td>
<td>-9.5%</td>
<td>-10.0%</td>
</tr>
<tr>
<td>Compress Kernel (-9)</td>
<td>35.03</td>
<td>501.6</td>
<td>-2.9%</td>
<td>-5.8%</td>
</tr>
<tr>
<td>Decompress Kernel</td>
<td>3.49</td>
<td>50.2</td>
<td>-0.7%</td>
<td>-1.6%</td>
</tr>
<tr>
<td>Method</td>
<td>Arch − Time</td>
<td>Power</td>
<td>CachyOS − Time</td>
<td>Power</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------</td>
<td>-------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>Compress Kernel (−best)</td>
<td>44.60</td>
<td>642.7</td>
<td>1.6%</td>
<td>10.5%</td>
</tr>
<tr>
<td>Compress Kernel (-5)</td>
<td>9.70</td>
<td>141.7</td>
<td>2.9%</td>
<td>10.2%</td>
</tr>
<tr>
<td>Decompress Kernel</td>
<td>0.73</td>
<td>12.1</td>
<td>-5.4%</td>
<td>-2.9%</td>
</tr>
<tr>
<td></td>
<td>Arch – Time</td>
<td>Power</td>
<td>CachyOS – Time</td>
<td>Power</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>-------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>Pybench</td>
<td>1472</td>
<td>290</td>
<td>3%</td>
<td>3.6%</td>
</tr>
</tbody>
</table>

My understanding is that the CachyOS package also includes BOLT optimization on top of `x86-64-v3` (which is said by upstream to improve performance by a % or two on such a benchmark). Even with additional optimizations it still lagged behind performance wise.
<table>
<thead>
<tr>
<th>Arch</th>
<th>Power</th>
<th>CachyOS Time</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-benchmark-25</td>
<td>39.41</td>
<td>2992.5</td>
<td>0%</td>
</tr>
</tbody>
</table>

CachyOS doesn’t rebuild the `r` package, but it does rebuild blas and lapack. According to `perf`, the test spends 86% of the time in `blas`, yet we still see no benefit. But, the proper way to actually improve performance here is to use `openblas` instead.
<table>
<thead>
<tr>
<th></th>
<th>Arch – Time</th>
<th>Power</th>
<th>CachyOS – Time</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decode</td>
<td>2.22</td>
<td>33.6</td>
<td>-10.7%</td>
<td>-8.4%</td>
</tr>
<tr>
<td>Encode (-b 128)</td>
<td>6.48</td>
<td>99.2</td>
<td>-20.8%</td>
<td>-19.5%</td>
</tr>
<tr>
<td>Encode (-q 10)</td>
<td>8.48</td>
<td>129.4</td>
<td>-14.7%</td>
<td>-12.8%</td>
</tr>
<tr>
<td></td>
<td>Arch – Time</td>
<td>Power</td>
<td>CachyOS – Time</td>
<td>Power</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
<td>-------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>Compress Kernel (-3 -T1)</td>
<td>75.03</td>
<td>1149.5</td>
<td>0.6%</td>
<td>0.7%</td>
</tr>
<tr>
<td>Compress Kernel (-9 -T4)</td>
<td>95.63</td>
<td>2638.2</td>
<td>1.2%</td>
<td>8.9%</td>
</tr>
<tr>
<td>Decompress Kernel</td>
<td>5.76</td>
<td>88.3</td>
<td>0.6%</td>
<td>1.3%</td>
</tr>
<tr>
<td></td>
<td>Arch - Time</td>
<td>Power</td>
<td>CachyOS - Time</td>
<td>Power</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>-------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>Compress Kernel (-19 - T4)</td>
<td>77.64</td>
<td>2443.8</td>
<td>-4.4%</td>
<td>-5.7%</td>
</tr>
<tr>
<td>Compress Kernel (-8 - T1)</td>
<td>11.09</td>
<td>181.8</td>
<td>-5.9%</td>
<td>-2.5%</td>
</tr>
<tr>
<td>Decompress Kernel</td>
<td>0.99</td>
<td>16.4</td>
<td>-15.7%</td>
<td>-0.2%</td>
</tr>
</tbody>
</table>
A Mixed Bag, But a Win Overall

Optimized packages can provide considerable advantages over their generic x86-64 counterparts for many packages. But where software doesn't see much benefit from these newer instructions, we are often left with worse performance and higher power consumption. This makes it a complex question over whether it's worthwhile to build every package with greater optimizations.

![Power vs Performance](image)

This graph shows that overall the winners were bigger than the regressions. However, this post was intended to be more about x86-64-v3, but some quick tests (which requires further analysis) suggest that CachyOS using -O3 is what's actually responsible for some of the larger gains rather than x86-64-v3.
Thanks

Q & A