# Bandwidth Performance Analysis of IPC Mechanisms

Akira Kawata

# IPC (Inter Process Communication) Mechanisms on Linux

- TCP via localhost
- Unix Domain Socket
- PIPE (See `man 2 pipe`)
- FIFO (named pipe. See `man 3 mkfifo`)
- POSIX message queue (See `man 3 mq_open`)
- mmap (See MAP_SHARED section in `man 2 mmap`)
- POSIX shared memory (See `man 3 shm_open`)

# Dissatisfaction with Existing Benchmarks

- lmbench
  - apt で入るパッケージがうまく動かない pull/32
  - 一部の測定結果がおかしい
- perf bench
  - memcpy の帯域しか計れない
- iperf3
  - TCP の帯域しか計れない
- そもそも自分で一回使ってみないと使える気持ちにならない

# akbench

- 作りました
  - https://github.com/akawashiro/akbench

```
$ git clone https://github.com/akawashiro/akbench.git
$ cd akbench
$ cmake -S . -B build -D CMAKE_CXX_COMPILER=clang++
$ cmake --build build
$ ./build/akbench/akbench bandwidth_all
```

# Bandwidth results on my computer

```
$ ./build/akbench/akbench bandwidth_all
bandwidth_memcpy: 18.098 ± 0.030 GiByte/sec
bandwidth_memcpy_mt (1 threads): 18.141 ± 0.368 GiByte/sec
bandwidth_memcpy_mt (2 threads): 18.649 ± 0.217 GiByte/sec
bandwidth_memcpy_mt (3 threads): 19.022 ± 0.515 GiByte/sec
bandwidth_memcpy_mt (4 threads): 18.641 ± 0.381 GiByte/sec
bandwidth_tcp: 5.779 ± 0.424 GiByte/sec
bandwidth_uds: 7.181 ± 0.212 GiByte/sec
bandwidth_pipe: 2.208 ± 0.035 GiByte/sec
bandwidth_fifo: 2.204 ± 0.024 GiByte/sec
bandwidth_mq: 1.811 ± 0.015 GiByte/sec
bandwidth_mmap: 10.869 ± 0.268 GiByte/sec
bandwidth_shm: 10.726 ± 0.196 GiByte/sec
```

# 測定結果の妥当性

# ベンチマークソフトウェアを自分で実装するのは不安

- 測り方を間違えていても数値は出る
- この数字あってるの?
- いろいろ比較しました

# perf mem bench memcpy

akbench で 18.0 GiByte/sec なのでまあまあ

```
$ /usr/lib/linux-tools/6.8.0-85-generic/perf bench mem memcpy --size
$((1<<30))
# Running 'mem/memcpy' benchmark:
# function 'default' (Default memcpy() provided by glibc)
# Copying 1073741824 bytes …
      18.756799 GB/sec
# function 'x86-64-unrolled' (unrolled memcpy() in arch/x86/lib/memcpy_64.S)
# Copying 1073741824 bytes …
      10.265465 GB/sec
# function 'x86-64-movsq' (movsq-based memcpy() in arch/x86/lib/memcpy_64.S)
# Copying 1073741824 bytes …
      18.771235 GB/sec
```

# iperf3

akbench で 5.77 GiByte/sec なのでまあまあ

```
$ iperf3 -c localhost --format g -bytes 1g
Connecting to host localhost, port 5201
[  5] local 127.0.0.1 port 59876 connected to 127.0.0.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  4.83 GBytes  4.82 GBytes/sec    0   6.56 MBytes
…
[  5]   9.00-10.00  sec  5.35 GBytes  5.35 GBytes/sec    0   6.56 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec  51.8 GBytes  5.18 GBytes/sec    0   sender
[  5]   0.00-10.00  sec  51.8 GBytes  5.18 GBytes/sec        receiver
```

# lmbench

lmbench の値はちょっと信用ならない

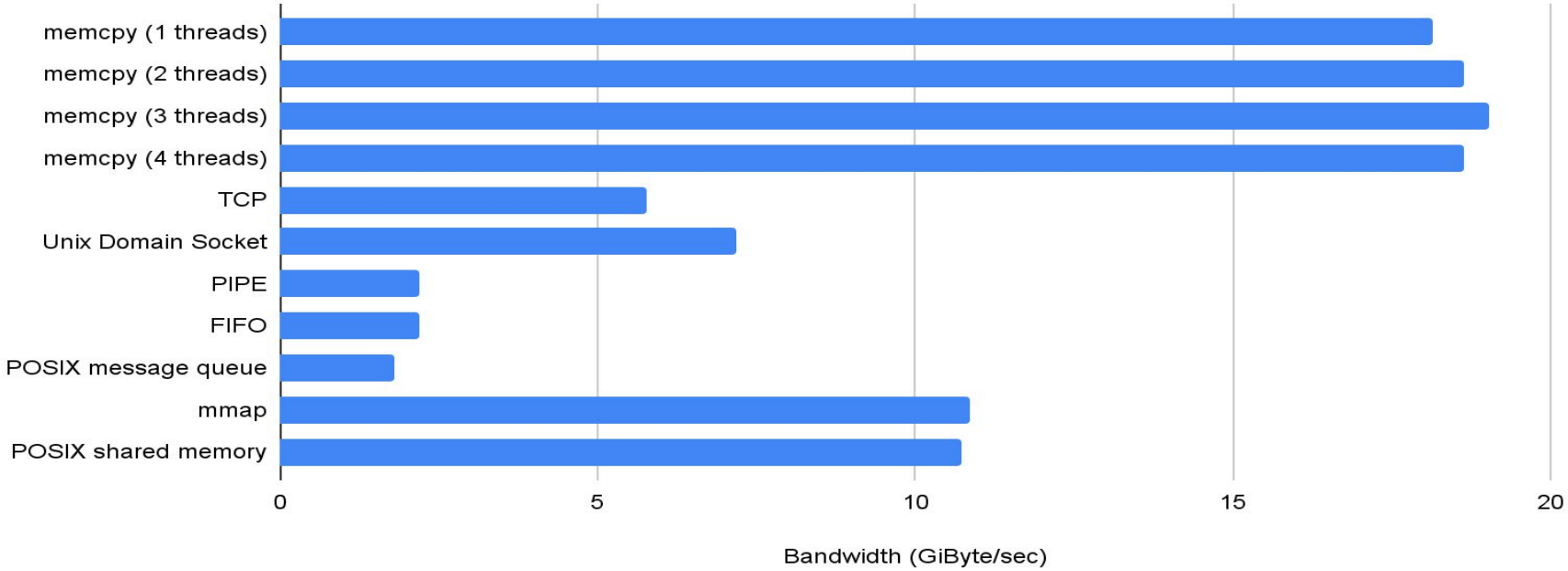
```
$ cd benchmark_comparison
$ ./run_lmbench.sh
bw_mem cp: 11.11 GiB/s # <= この値なんか怪しい。低すぎる
bw_pipe: 3.31 GiB/s    # <= これは妥当か?
bw_unix: 0.08 GiB/s    # <= この値なんか怪しい。低すぎる
```

# 測定結果の分析

# [再掲] Bandwidth results on my computer

```
$ ./build/akbench/akbench bandwidth_all
bandwidth_memcpy: 18.098 ± 0.030 GiByte/sec
bandwidth_memcpy_mt (1 threads): 18.141 ± 0.368 GiByte/sec
bandwidth_memcpy_mt (2 threads): 18.649 ± 0.217 GiByte/sec
bandwidth_memcpy_mt (3 threads): 19.022 ± 0.515 GiByte/sec
bandwidth_memcpy_mt (4 threads): 18.641 ± 0.381 GiByte/sec
bandwidth_tcp: 5.779 ± 0.424 GiByte/sec
bandwidth_uds: 7.181 ± 0.212 GiByte/sec
bandwidth_pipe: 2.208 ± 0.035 GiByte/sec
bandwidth_fifo: 2.204 ± 0.024 GiByte/sec
bandwidth_mq: 1.811 ± 0.015 GiByte/sec
bandwidth_mmap: 10.869 ± 0.268 GiByte/sec
bandwidth_shm: 10.726 ± 0.196 GiByte/sec
```

# Graph of bandwidth results on my computer



# memcpy performance on my computer

- Memory spec
  - DDR4 / clock: 3200MHz / width: 64 bits
  - 25.6 GiByte/sec (= 3200 MHz * 64 bits)
- memcpy の帯域が 18 GiByte/sec
  - perf mem bench でも同じぐらい出た
- 70% ぐらい出てる?
  - 読み書きが同じメモリに対して行われているとすると 140 %ということになるが...
- マルチスレッドで memcpy の速度が改善する?

# TCP vs UDS (Unix Domain Socket)

- UDS のほうが TCP のオーバーヘッドがない分速い
  - TCP が 5.7 GiByte/sec で UDS が 7.18 GiByte/sec

# PIPE と FIFO

- たぶんこの二つは内部的な実装が同じ
  - man 3 mkfifo に named pipe とあるぐらいなので

# POSIX message queue

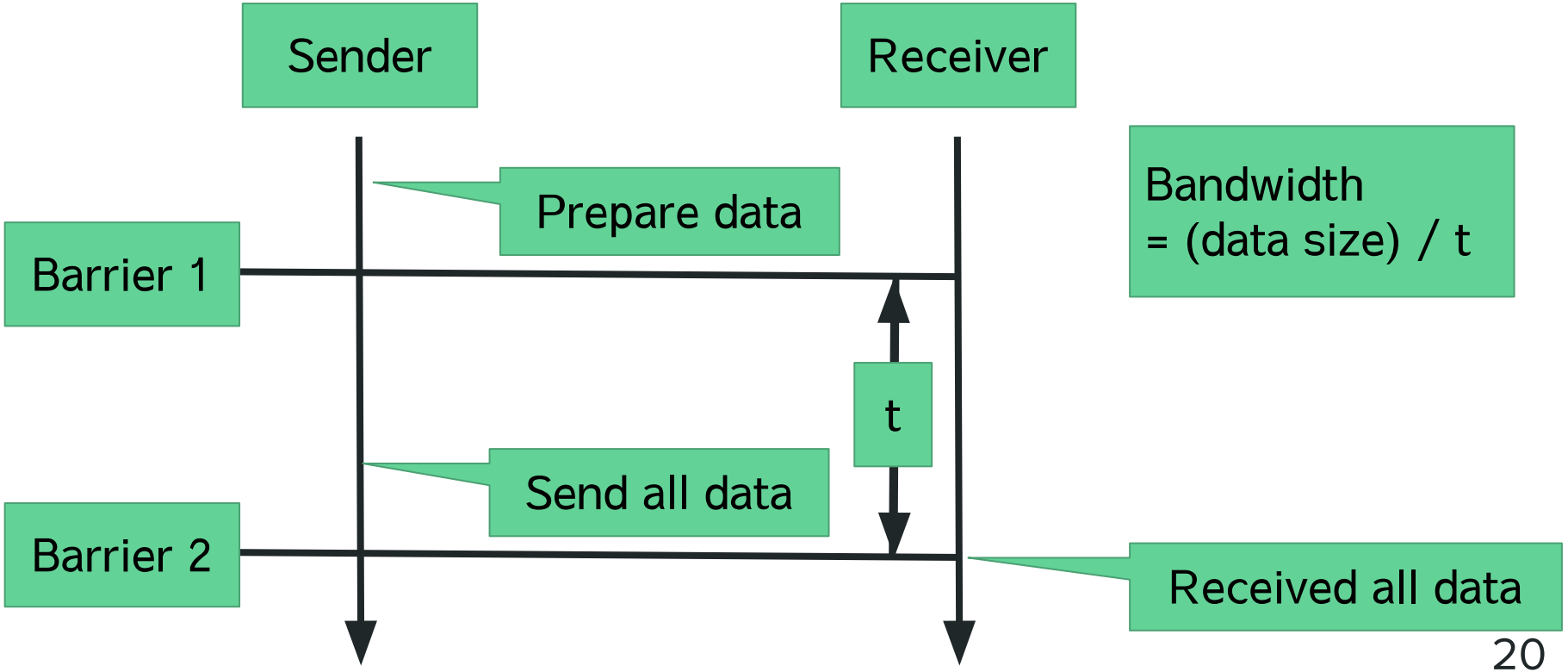
- 計測した IPC mechanism の中でダントツに遅い
  - 優先度がつけられるらしい
- 使うんですか...これ?

# mmap と shm

- プロセス間通信の中で最速
- mmap と shm はほとんど同じ性能が出る
- => MAP_SHARED と /dev/shm はおそらく内部的な実装が同じ

# ベンチマークの実装

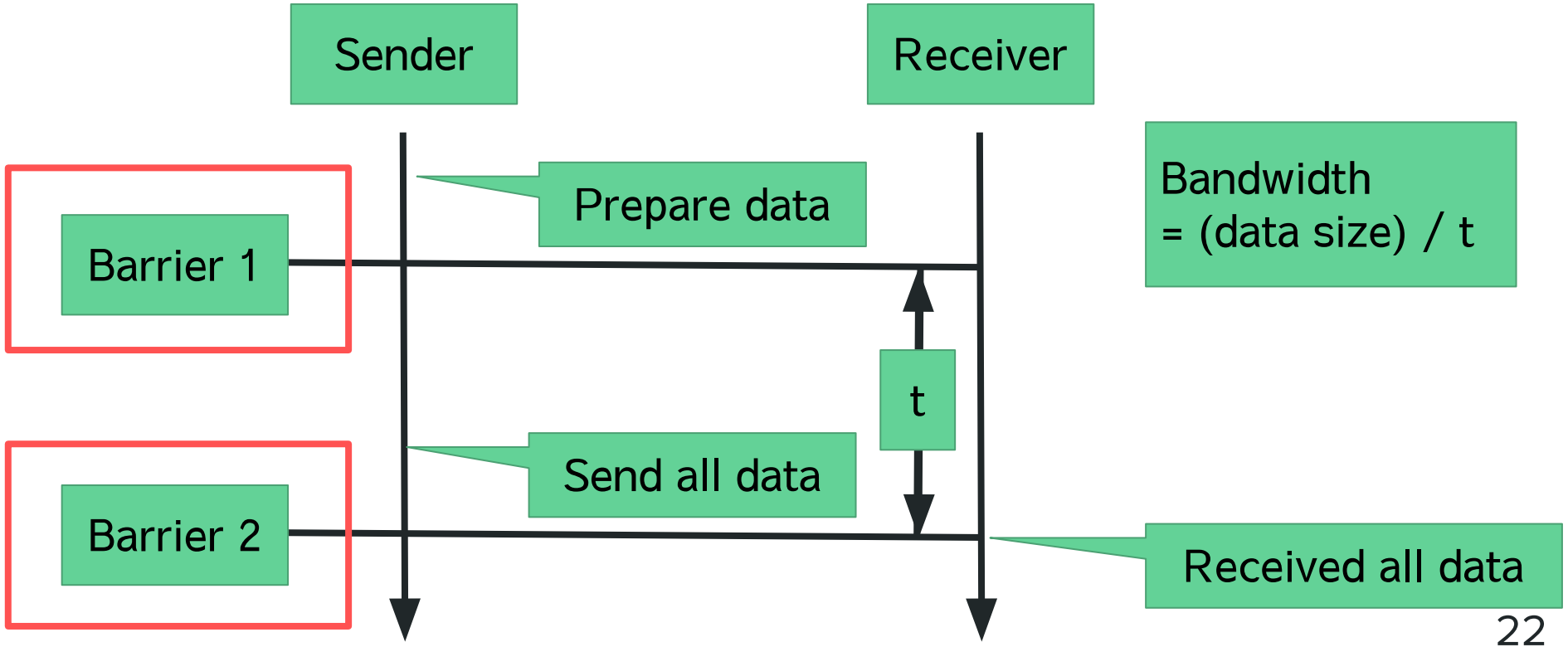
# How akbench calculates bandwidth?



# How akbench calculates bandwidth?

- 送信側で送り始めてから受信側で受信し終わるまでの時間を計測
- 送信側で送り始めた時間をバリアで同期
- ほんとは ping-pong で計ったほうがよさそう

# How akbench synchronize two process?



# How akbench synchronize two process?

- センス反転バリアを実装 (barrier.cc)
  - データ置き場: shared memory
  - 同期機構: セマフォ
  - std::atomic はプロセス間同期で利用できるかがわからず使っていない
    - C++ の規格にプロセスという概念がない
- 1 us ぐらいで同期できる
  - 帯域を計測するときはデータの送受信に 100ms ぐらいかかるようにして、同期のオーバーヘッドが影響が出ないようにしている

```
$ ./build/akbench/akbench latency_barrier
Barrier benchmark result: 1066.952 ± 342.502 ns
```

# Latency Performance Analysis of Inter Process/Thread Communication Mechanisms

Akira Kawata

# Latency results on my computer

```
$ ./build/akbench/akbench latency_all
latency_atomic: 30.603 ± 7.456 ns
latency_atomic_rel_acq: 28.425 ± 6.766 ns
latency_barrier: 1207.463 ± 501.146 ns
latency_condition_variable: 3154.424 ± 707.784 ns
latency_semaphore: 2962.199 ± 681.565 ns
latency_statfs: 1008.512 ± 225.512 ns
latency_fstatfs: 664.642 ± 148.630 ns
latency_getpid: 95.559 ± 21.436 ns
```

# lmbench

```
$ ./run_lmbench.sh
…
lat_syscall null: 132.60 ns
lat_syscall read: 218.80 ns
lat_syscall write: 196.80 ns
lat_syscall stat: 824.30 ns
lat_syscall fstat: 316.10 ns
lat_syscall open: 1976.90 ns
lat_sem: 1220.70 ns
```

# Latency of synchronization mechanisms

- 上二つは OS を経由しないので 100 倍ぐらい早い
- release acquire をちゃんとしても 2.5 ns ぐらいしか変わらない
- latency_condition_variable と latency_semaphore はなんかおかしい
  - 3 倍ぐらいの値が出ている

```
$ ./build/akbench/akbench latency_all
latency_atomic: 30.603 ± 7.456 ns
latency_atomic_rel_acq: 28.425 ± 6.766 ns
latency_barrier: 1207.463 ± 501.146 ns
latency_condition_variable: 3154.424 ± 707.784 ns
latency_semaphore: 2962.199 ± 681.565 ns
```

# Latency of syscall

- 一番軽い syscall 一回で 100 ns ぐらい
  - lmbench と比較しても妥当

```
$ ./build/akbench/akbench latency_all
latency_statfs: 1008.512 ± 225.512 ns
latency_fstatfs: 664.642 ± 148.630 ns
latency_getpid: 95.559 ± 21.436 ns
```