LLVM C Library for GPUs

Joseph Huber ***(joseph.huber@amd.com)***
LLVM Developer's Conference 2023

AMD
together we advance_

What is the LLVM C library for GPUs

Using libc for GPUs

Table of Contents

Building the GPU library

LLVM's libc GPU support *must* be built with an up-to-date `clang` compiler due to heavy reliance on `clang`'s GPU support. This can be done automatically using the `LLVM_ENABLE_RUNTIMES=libc` option. To enable libc for the GPU, enable the `LIBC_GPU_BUILD` option. By default, `libcgpu.a` will be built using every supported GPU architecture. To restrict the number of architectures build, either set `LIBC_GPU_ARCHITECTURES` to the list of desired architectures manually or use `native` to detect the GPUs on your system. A typical `cmake` configuration will look like this:

```
$> cd llvm-project  # The llvm-project checkout
$> mkdir build
$> cd build
$> cmake ../llvm -G Ninja                              \
   -DLLVM_ENABLE_PROJECTS="clang;lld;compiler-rt"      \
   -DLLVM_ENABLE_RUNTIMES="libc;openmp"                \
   -DCMAKE_BUILD_TYPE=<Debug|Release>   \ # Select build type
   -DLIBC_GPU_BUILD=ON                  \ # Build in GPU mode
   -DLIBC_GPU_ARCHITECTURES=all         \ # Build all supported architectures
   -DCMAKE_INSTALL_PREFIX=<PATH>        \ # Where 'libcgpu.a' will live
$> ninja install
```

- Build of the LLVM C library targeting GPUs
  - https://libc.llvm.org/gpu
  - **-DLIBC_GPU_BUILD=ON**
- Supports AMD and NVIDIA GPUs

AMD
together we advance_

Why write a C library for the GPU

- Initially wanted a portable implementation of **printf**
  - Might as well do everything else while we're at it
- Trivially port CPU applications and tests to the GPU
- Portable GPU math functions in **libmgpu.a**
- Basis for more libraries, i.a. **libc++**

AMD
together we advance_

LLVM C Library — Language Support

- Exported as **libcgpu.a** and **libmgpu.a**
- Compatible with OpenMP, C++*, CUDA*, HIP*
- Support for most common **libc** functions
  - https://libc.llvm.org/gpu/support.html

stdlib.h

Function Name	Available	RPC Required
abs	✅	
atoi	✅	
atof	✅	
atol	✅	
atoll	✅	
exit	✅	✅
abort	✅	✅
labs	✅	
llabs	✅	
div	✅	
ldiv	✅	
lldiv	✅	
bsearch	✅	
qsort	✅	
qsort_r	✅	
strtod	✅	
strtof	✅	
strtol	✅	
strtold	✅	
strtoll	✅	
strtoul	✅	
strtoull	✅	

What’s Working — Compiling and Running

```cpp
#include <omp.h>
#include <stdio.h>

int main() {
#pragma omp target
#pragma omp parallel num_threads(4)
  fprintf(stdout, "%s%d\n",
          "Thread id: ", omp_get_thread_num());
}
```

```
$ clang++ openmp.cpp -fopenmp --offload-arch=native \
          -lcgpu
$ ./a.out
Thread id: 0
Thread id: 1
Thread id: 2
Thread id: 3
```

```cpp
#include <stdio.h>

int main() {
  fprintf(stdout, "%s%d\n", "Thread id: ",
          __builtin_amdgcn_workitem_id_x());
}
```

```
$ clang++ direct.cpp --target=amdgcn-amd-amdhsa \
          -mcpu=native -lc crt1.o
$ ./amdhsa_loader --threads 4 a.out
Thread id: 0
Thread id: 1
Thread id: 2
Thread id: 3
```

What’s Working — Build Bots

LLVM Staging — Builders / openmp-offload-libc-amdgpu-runtime

Build requests | Build times | Success Rate

None

Builds:

#	Started At	Duration	Owners
176	23 minutes ago	8 minutes	Peter Kasting <pkasting@chromium.org>
175	31 minutes ago	8 minutes	Mark de Wever <koraq@xs4all.nl>
174	42 minutes ago	11 minutes	Simon Pilgrim <llvm-dev@redking.me.uk>
173	an hour ago	8 minutes	Jay Foad <jay.foad@amd.com>
172	2 hours ago	9 minutes	Jay Foad <jay.foad@amd.com>
171	2 hours ago	8 minutes	Michael Liao <michael.hliao@gmail.com>
170	3 hours ago	8 minutes	Simon Pilgrim <llvm-dev@redking.me.uk>

AMDGPU builder targeting **gfx906** architecture

LLVM Buildbot — Builders / clang-cuda-t4

Build requests | Build times | Success Rate

None

Builds:

#	Started At	Duration	Owners
47783	37 minutes ago	7 minutes	Peter Kasting <pkasting@chromium.org>
47782	an hour ago	6 minutes	Mark de Wever <koraq@xs4all.nl>
47781	an hour ago	15 minutes	Simon Pilgrim <llvm-dev@redking.me.uk>
47780	2 hours ago	13 minutes	Jay Foad <jay.foad@amd.com>
47779	2 hours ago	6 minutes	Michael Liao <michael.hliao@gmail.com>
47778	3 hours ago	8 minutes	Simon Pilgrim <llvm-dev@redking.me.uk>
47777	3 hours ago	6 minutes	Alexey Lapshin <a.v.lapshin@mail.ru>

NVIDIA builder targeting the **sm_75** architecture

LLVM Buildbot — Builders / clang-cuda-p4

Build requests | Build times | Success Rate

None

Builds:

#	Started At	Duration	Owners
47966	34 minutes ago	7 minutes	Peter Kasting <pkasting@chromium.org>
47965	an hour ago	7 minutes	Mark de Wever <koraq@xs4all.nl>
47964	an hour ago	15 minutes	Simon Pilgrim <llvm-dev@redking.me.uk>
47963	2 hours ago	14 minutes	Jay Foad <jay.foad@amd.com>
47962	2 hours ago	7 minutes	Michael Liao <michael.hliao@gmail.com>
47961	3 hours ago	8 minutes	Simon Pilgrim <llvm-dev@redking.me.uk>
47960	3 hours ago	6 minutes	Alexey Lapshin <a.v.lapshin@mail.ru>

NVIDIA builder targeting the **sm_60** architecture

The LLVM C Library — Overview

- Written entirely in freestanding C++
- Designed to be easily decomposed and ported
  - Generates custom system headers (e.g. string.h)
  - Defines which functions are supported (e.g. memcpy)
  - Configured through function specifications (e.g. the C standard)

```
set(TARGET_PUBLIC_HEADERS
    libc.include.ctype
    libc.include.string
    libc.include.fenv
    libc.include.errno
    libc.include.stdlib
)
```

```
set(TARGET_LIBC_ENTRYPOINTS
    # ctype.h entrypoints
    libc.src.ctype.isalnum
    libc.src.ctype.isspace
    …
    # string.h entrypoints
    libc.src.string.memccpy
    libc.src.string.memcmp
    libc.src.string.memcpy
    …
)
```

```
HeaderSpec CType = HeaderSpec<
  "ctype.h",
  [], // Macros
  [], // Types
  [
    FunctionSpec<
      "isalnum",
      RetValSpec<IntType>,
      [ArgSpec<IntType>]
    >,
  ]
>;
```

The LLVM C Library — Targeting GPUs

```cpp
#include "src/string/strcmp.h"
#include "src/__support/common.h"

namespace LIBC_NAMESPACE {

template <typename Comp>
constexpr static int strcmp_implementation(
  const char *left, const char *right, Comp &&comp) {
  for (; *left && !comp(*left, *right); ++left, ++right)
    ;
  return comp(*reinterpret_cast<unsigned char *>(left),
              *reinterpret_cast<unsigned char *>(right));
}

LLVM_LIBC_FUNCTION(int, strcmp, (const char *left,
                   const char *right)) {
  auto comp = [](char l, char r) -> int { return l - r; };
  return strcmp_implementation(left, right, comp);
}

} // namespace LIBC_NAMESPACE
```

- GPUs are treated as regular targets
  - **--target=amdgcn-amd-amdhsa -mcpu=gfx90a**
- Architecture specific operations handled with macros and builtin functions
  - **__AMDGPU__** and **__NVPTX__**
- Packaged as a “bundled” static library (just link it!)

The LLVM C Library — Targeting GPUs

C++ source #1

```cpp
extern "C" int isdigit(unsigned c) {
  return (c - '0') < 10;
}
```

x86-64 clang (assertions trunk) (Editor #1)

--target=amdgcn-amd-amdhsa -mcpu=gfx1030 -nogpulib -S -g0 -O3

```
isdigit:                                ; @isdigit
        s_waitcnt vmcnt(0) expcnt(0) lgkmcnt(0)
        v_subrev_nc_u32_e32 v0, 48, v0
        v_cmp_gt_u32_e32 vcc_lo, 10, v0
        v_cndmask_b32_e64 v0, 0, 1, vcc_lo
        s_setpc_b64 s[30:31]
llvm.amdgcn.abi.version:
        .long   400                             ; 0x190

---
amdhsa.target:   amdgcn-amd-amdhsa--gfx1030
  - 1
```

Output (0/1) x86-64 clang (assertions trunk) - 1630ms (1309B) ~34 lines filtered

x86-64 clang (assertions trunk) (Editor #1)

--target=amdgcn-amd-amdhsa -mcpu=gfx1030 -nogpulib -S -g0 -O3 -emit-llvm

```llvm
@llvm.amdgcn.abi.version = weak_odr hidden local_unnamed_addr addrspace(4) constant i32

define hidden i32 @isdigit(i32 noundef %c) local_unnamed_addr #0 {
entry:
  %sub = add i32 %c, -48
  %cmp = icmp ult i32 %sub, 10
  %conv = zext i1 %cmp to i32
  ret i32 %conv
}
```

The LLVM C Library — GPU Loader Utility and Startup

- Write a loader utility to launch the GPU program
  - **amdhsa_loader** and **nvptx_loader**
- Standard **libc** implementations use a startup object (i.e. **crt1.o**)
  - Just write one for the GPU
- Export kernels that handle global ctors / dtors and call the **main** function

```
void call_init_array_callbacks(int argc, char **argv, char **env) {
  // Call global constructors.
}

void call_fini_array_callbacks() {
  // Call global destructors.
}

extern "C" {
[[gnu::visibility("protected"), clang::amdgpu_kernel]] void
_begin(int argc, char **argv, char **env, void *in, void *out) {
  atexit(&call_fini_array_callbacks);
  call_init_array_callbacks(argc, argv, env);
}

[[gnu::visibility("protected"), clang::amdgpu_kernel]] void
_start(int argc, char **argv, char **envp, int *ret) {
  __atomic_fetch_or(ret, main(argc, argv, envp),
                    __ATOMIC_RELAXED);
}

[[gnu::visibility("protected"), clang::amdgpu_kernel]] void
_end(int retval) {
  exit(retval);
}
}
```

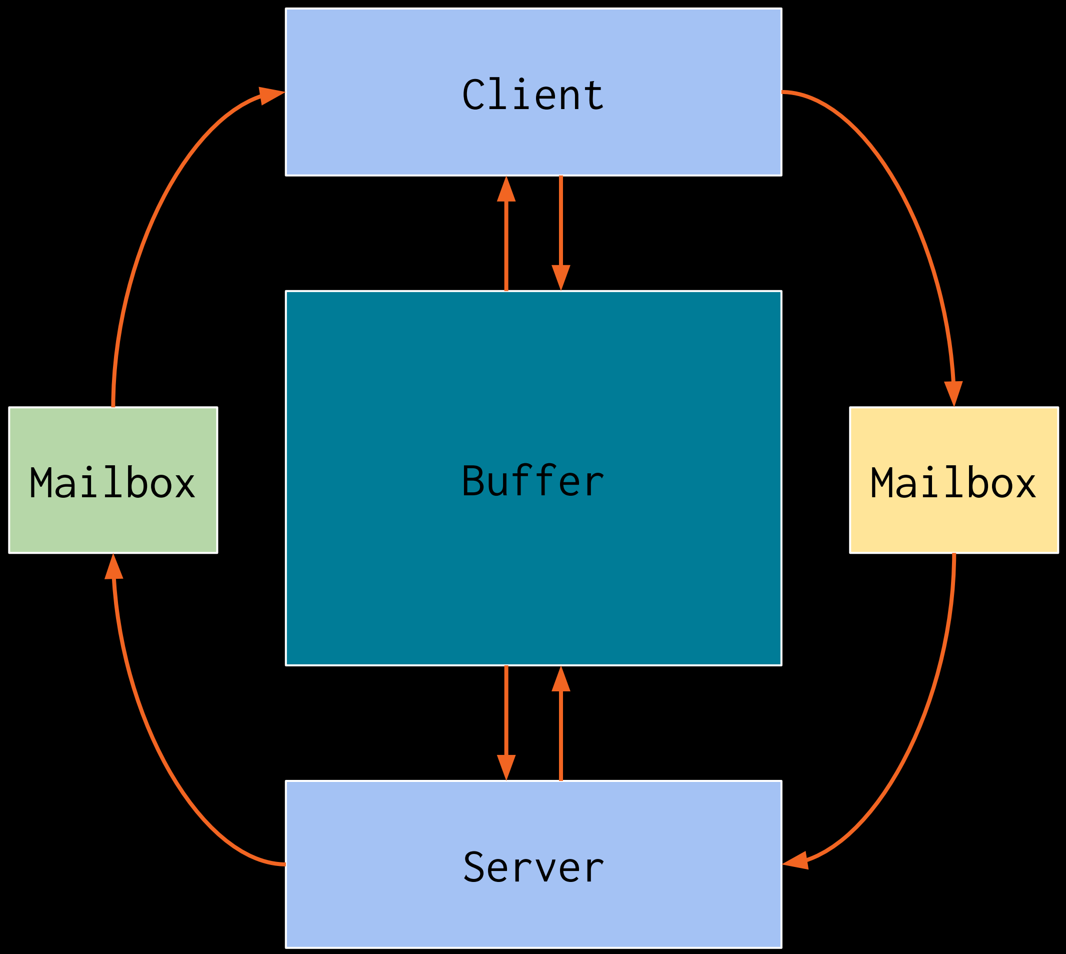
Now we can run anything on the GPU right?

The LLVM C Library — Syscalls

- Some functions require intervention from the operating system
  - E.g. `exit`, `printf`, or `malloc`
- The GPU doesn't have a usable operating system
  - Treat the host machine as the operating system
- Implement **Remote Procedure Calls** to function as syscalls

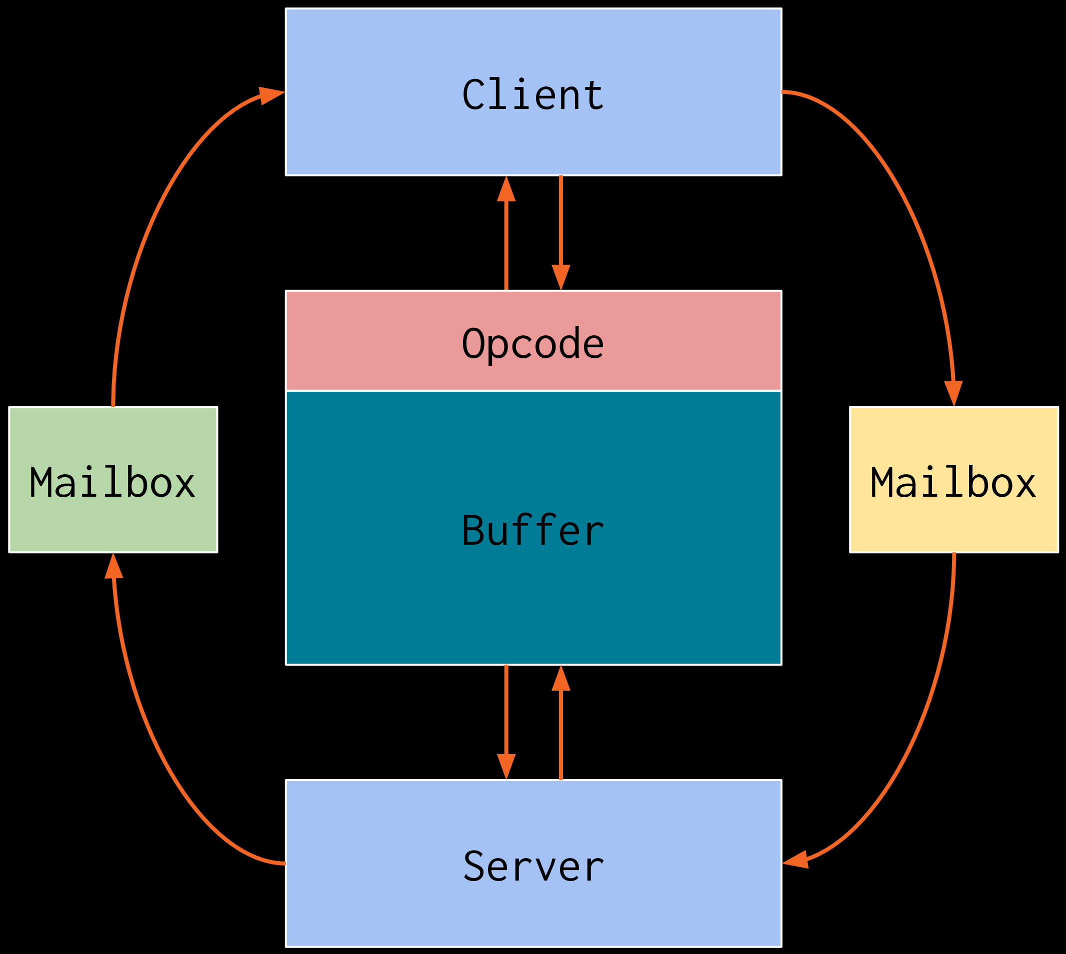
Remote Procedure Calls — Implementation

- GPUs support device-accessible host memory
  - E.g. hipMallocHost or cudaMallocHost
- Atomically swap ownership of a fixed size buffer between a client and server
  - Each process has a write-only outbox and a read-only inbox to indicate ownership of the buffer
- Exposes some primitive operations
  - Wait for ownership
  - Use the buffer
  - Give away ownership



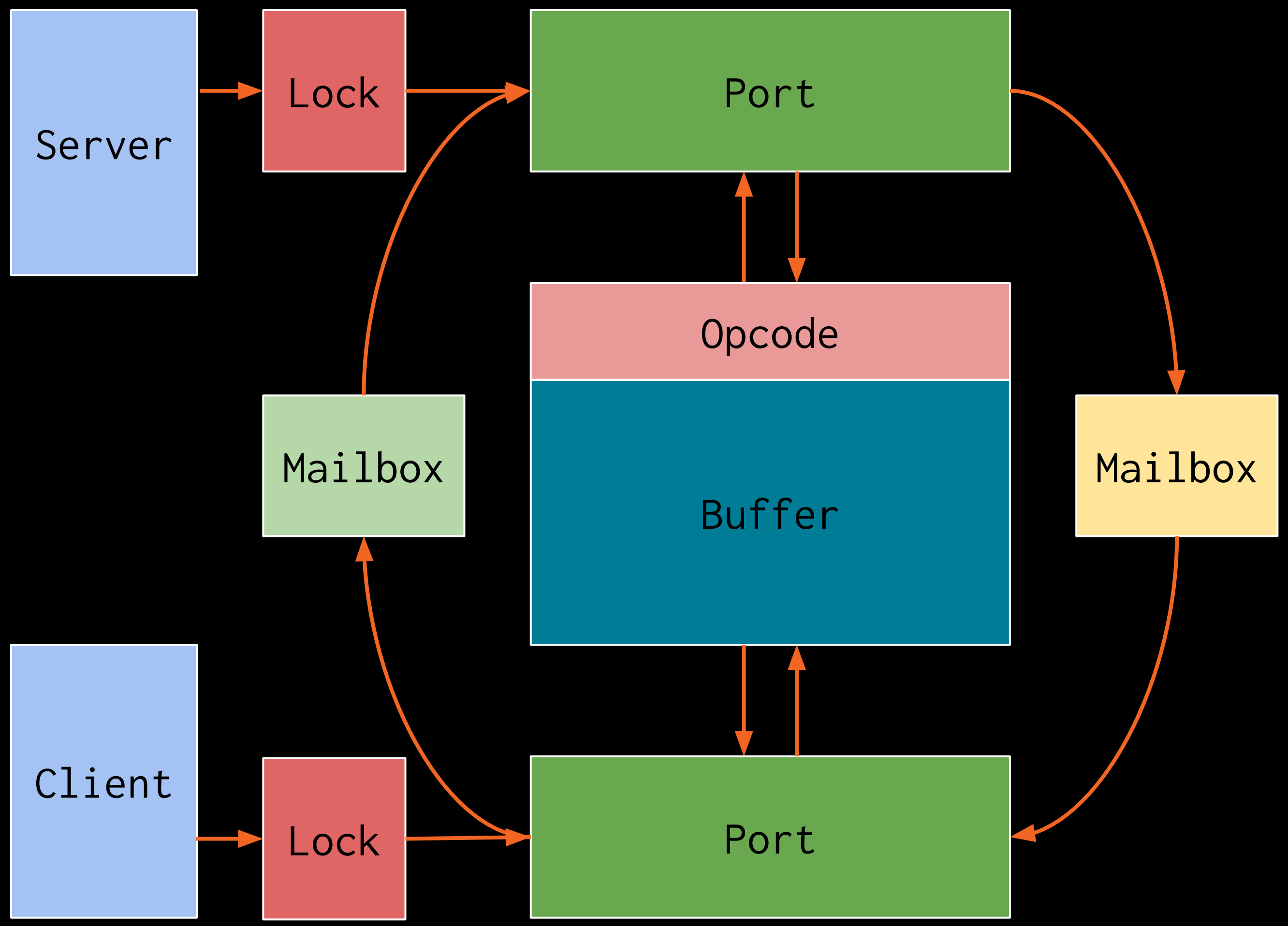
Remote Procedure Calls — Implementation

- Most GPUs support unified memory addressing
  - E.g. hipMallocHost or cudaMallocHost
- Atomically swap ownership of a fixed size buffer between a client and server
  - Each process has a write-only outbox and a read-only inbox to indicate ownership of the buffer
- Exposes some primitive operations
  - Wait for ownership
  - Use the buffer
  - Give away ownership
- Add a header for the desired **“syscall”**



Remote Procedure Calls — Handling multiple threads

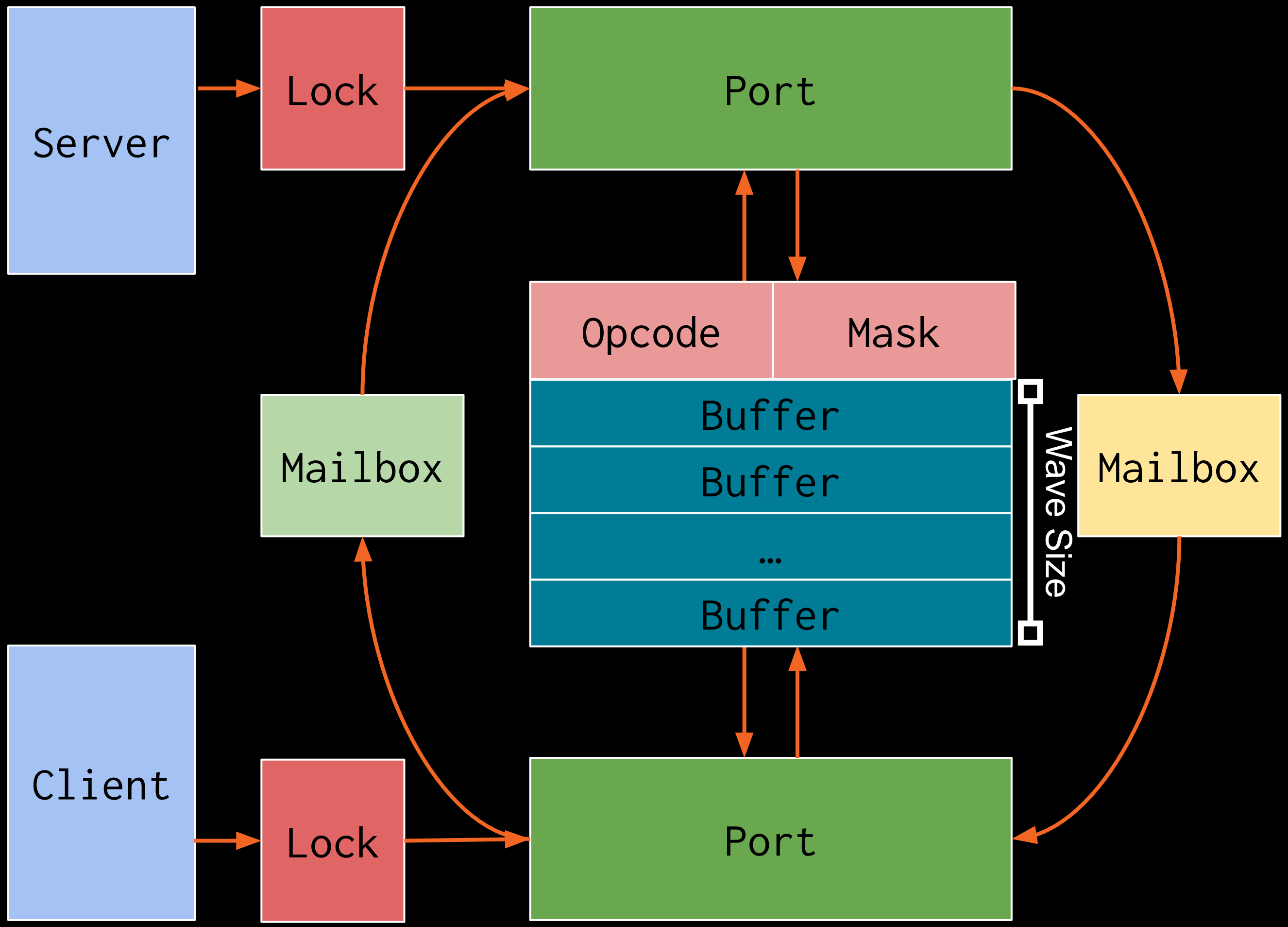
- Abstract access to the buffer into a **port**
- Provide mutual exclusion on the port using a **test** and **set** lock
- We can open a port if the process owns the buffer and the lock is available



AMD together we advance_

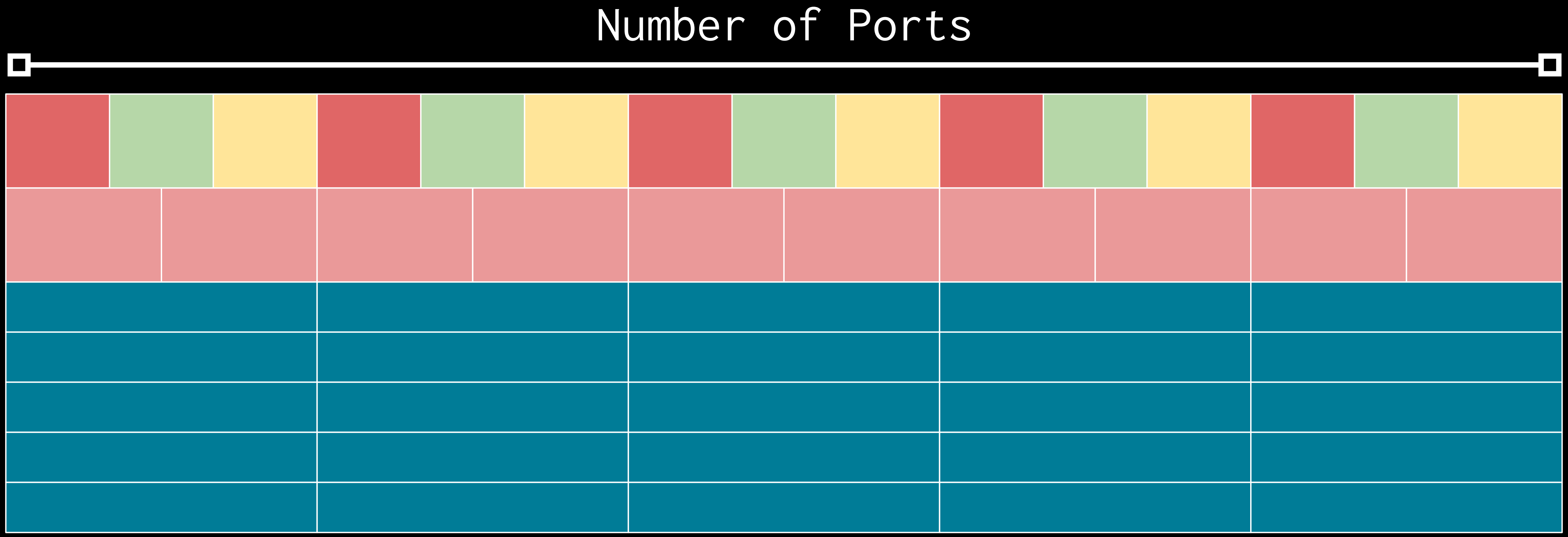
Remote Procedure Calls — GPU Considerations

- The smallest unit of independent parallelism on GPUs is not a thread
  - SIMD execution on a warp / wave
- The interface needs to handle a whole warp or wavefront at a time
  - Supports partial / masked usage



Remote Procedure Calls — Implementation

- Provide multiple ports to increase concurrent access
  - Required to prevent deadlocks on some GPU hardware



The LLVM C Library — Remote Procedure Calls

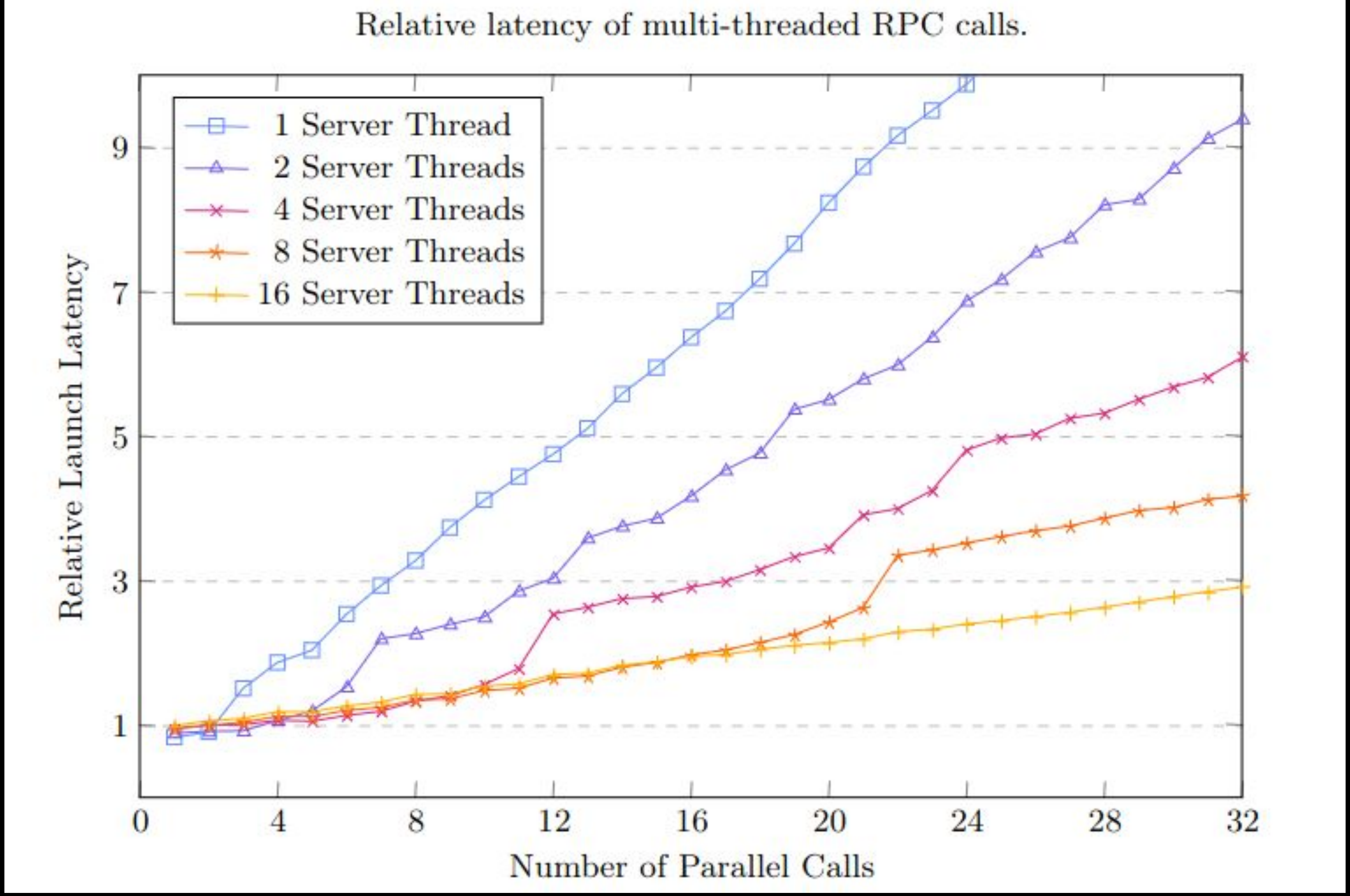
- Simplify this interface into arbitrary* **send** and **recv** of packets
- Requires a server on the host listening to the ports
- Provides the ability to call host functions from the GPU

```
#include "rpc.h"
// Running on GPU.
uint64_t increment_on_cpu(uint64_t count) {
  rpc::Client::Port port = rpc::client.open<RPC_INC>();
  port.send([=](uint64_t *buffer) { *buffer = count; });
  port.recv([&](uint64_t *buffer) { count = *buffer; });
  port.close();
  return count;
}

// Running on CPU.
int rpc_server() {
  rpc::Server::Port port = rpc::server.open();
  switch (port.opcode) {
  case RPC_INC: {
    port.recv([&](uint64_t *buffer) { *buffer += 1; });
    port.send([](uint64_t *) { /* no-op */ });
  } break;
  default:
    break;
  }
  port.close();
}
```

AMD
together we advance_

Remote Procedure Calls — Overhead



- Comparing the latency of a **no-op** function call on the host via RPC versus a kernel launch
  - Calling an RPC function is roughly equivalent to launching a kernel

AMD
together we advance_

```
#include "src/string/strcmp.h"
#include "test/UnitTest/Test.h"

TEST(LlvmLibcStrCmpTest,
     CapitalizedLetterShouldNotBeEqual) {
  const char *s1 = "abcd";
  const char *s2 = "abCd";
  int result = __llvm_libc::strcmp(s1, s2);
  // 'c' - 'C' = 32.
  ASSERT_EQ(result, 32);

  // Verify operands reversed.
  result = __llvm_libc::strcmp(s2, s1);
  // 'C' - 'c' = -32.
  ASSERT_EQ(result, -32);
}
```

The LLVM C Library — Testing

- The existing LLVM C library tests run on a CPU in a self-hosted environment
- Execute these tests directly on the GPU as if we were cross compiling
- Running 125 GPU tests across three buildbots
  - Looking into running parts of the LLVM test suite on the GPU as well

The LLVM C Library — Pain Points

- Currently cannot support **thread_local** keywords
  - Implementing **errno** or **rand** is difficult without it
- Linking **libc** with LTO is problematic
  - Functions cannot be inlined without **-fno-builtin**
- CMake cannot build the CPU **libc** and GPU **libc** at the same time
- The **libc** headers and host headers need to match and cooperate
- The NVIDIA toolchain
  - No static library support in **nvlink**
  - Cannot emit variables to sections, the **ptx** keyword **section** only works in debug mode
- Multi-architecture support should be done with a single bitcode file
  - We currently just build the entire library for each supported architecture

LibC for GPUs — Conclusion

- Built the LLVM C library targeting GPU
  - Compiling freestanding C++ for the GPU works very well and is portable
- Created **libcgpu.a** and **libmgpu.a** that is fully integrated into OpenMP offloading
  - Using with CUDA / HIP is functional but opt-in
- Parallel and extensible RPC interface for host execution
- Available now upstream and tested via buildbots
- Can compile and run standard C++ directly on the GPU

Thanks to

- Jon Chesterfield
- Johannes Doerfert
- Brian Sumner
- Tian Shilei
- Siva Chandra
- Tue Ly
- Michael Jones