That's right, the same C++ STL asynchronous parallel code runs on CPUs & GPUs

Muhammad Haseeb, Weile Wei, Jack Deslippe, and Brandon Cook

Lawrence Berkeley National Laboratory, Berkeley, CA, USA

BERKELEY LAB

NERSC National Energy Research Scientific Computing Center

U.S. DEPARTMENT OF ENERGY Office of Science

Motivation

Various programming models are used to accelerate C++ HPC applications on heterogeneous supercomputers.

This leads to many challenges in software performance, code portability, and programming productivity.

Introduction

std::execution (*or* ***stdexec***) -- the proposed C++26 standard feature -- provides an asynchronous parallel framework enabling running standard C++ code on CPUs & GPUs with minimal modifications.

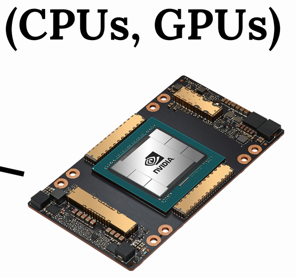
We leverage ***stdexec***, ***stdpar*** & modern C++ to implement & evaluate multiple HPC scientific apps on the ***Perlmutter*** supercomputer.

std::execution components

1) ***schedulers*** - obtained from execution contexts

```
scheduler c = cpu.schedule();
scheduler g = gpu.schedule();
```

execution contexts (CPUs, GPUs)



2) ***senders*** - send the ***composed*** work to scheduler **g**

```
sender auto s = schedule(g)
```

```
| bulk (N, par_alg1())
| bulk (N, par_alg2())
| transfer (c)
| then (serial_alg1());
```

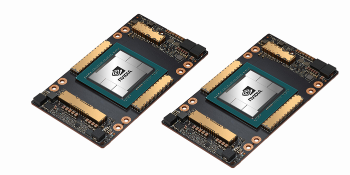
3) ***composable algorithms*** - compose the algorithmic task graph (DAG) using pipe symbol (**|**)

4) ***receivers*** - wait and receive the execution results for sender **s**

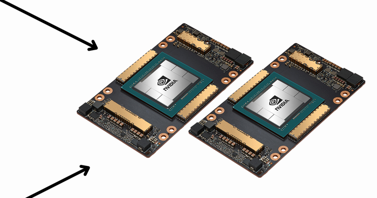
```
auto [r] = sync_wait(s).value();
```

App 1: AMReX 2D Stencil

```
// nvexec multi gpu stream context and scheduler
nvexec::multi_gpu_stream_context ctx;
auto gpus = ctx.get_scheduler();
// std::spans to ensure trivially copyable
std::span phi_old, phi_new;
```



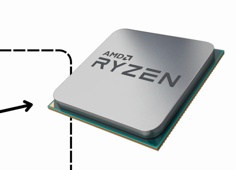
```
// send initialize phi_old to gpus
sender auto s1 = transfer_just (gpus, phi_old)
  | bulk (phi_old, init);
sync_wait(s1);
```

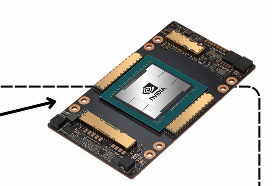


```
for (i in nsteps) {
  // send heat equation loop to gpus
  sender auto s2 = transfer_just (gpus, phi_old, phi_new)
    | bulk (phi_old, phi_new, fill_boundary(old, new))
    | bulk (phi_old, phi_new, heat_equation(old, new))
    | bulk (phi_old, phi_new, parallel_copy(old, new)); }
```

App 2: Recursive 1D Stecil

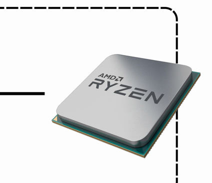
```
// recursive stencil computation
auto stencil (nsteps) -> any_space_sender {
  // initialize & return phi on cpu
  if (nsteps == 0) {
    mdspan phi = std::for_each(stdexec::seq, init);
    return just (phi); }

  return just(nsteps - 1)
    | let_value((step) { return stencil(step); })
    | then((auto phi_old) {

    // stencil on GPU using stdpar
    mdspan phi_new;
    std::for_each(stdexec::par, jacobi(phi_new, phi_old));
    return phi_new; }); }
```



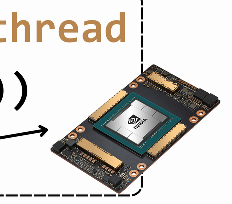


App 3: ADEPT Drivers

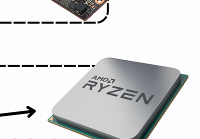
```
// thread pool context and scheduler
static_thread_pool ctx {N};
scheduler cpus = ctx.get_scheduler();
```



```
// compose adept pipeline - one CUDA kernel per thread
sender adept = schedule(cpus) | then(init_adept())
  | bulk(N, partition_launch_cuda(N));
```



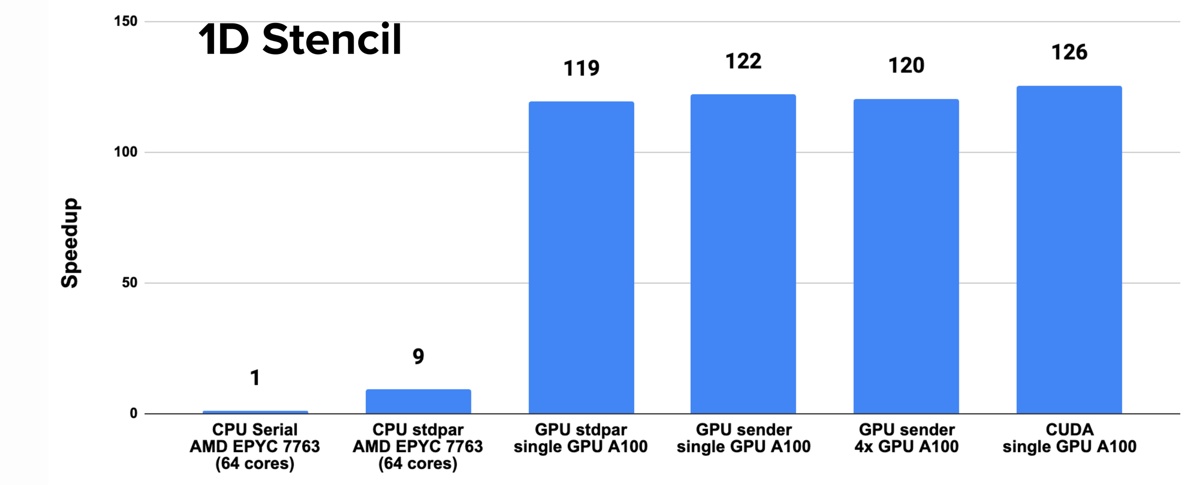
```
// do other async work while adept is running
sender otherwork = schedule(cpus) | then(otherwork());
```

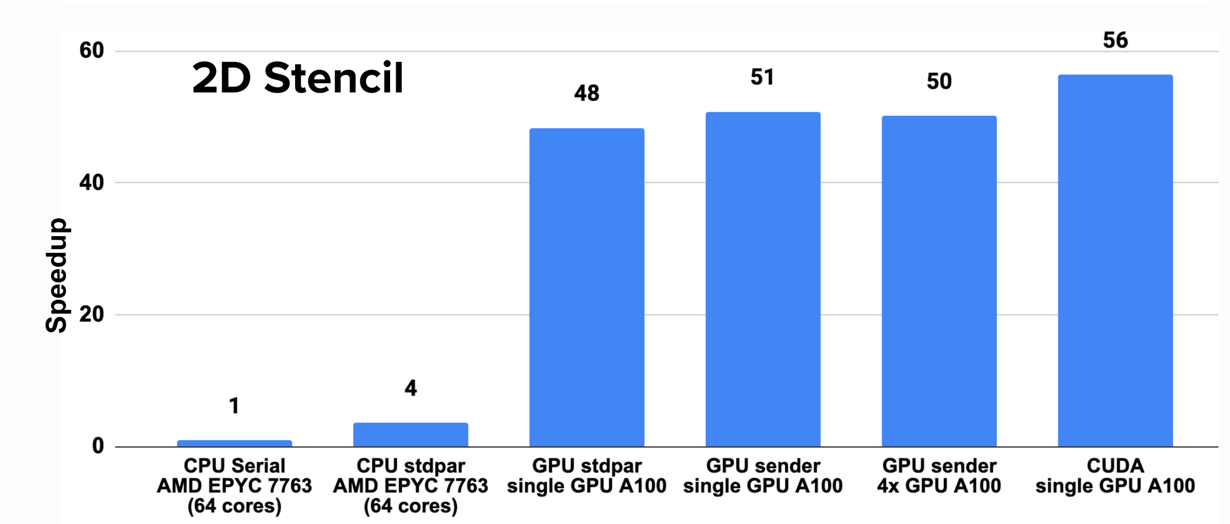


Preliminary Results

Experimental Setup: Perlmutter GPU (NVIDIA A100 SXM), AMD EPYC 7763, NVHPC/23.7, GCC/12.2, NVIDIA/stdexec

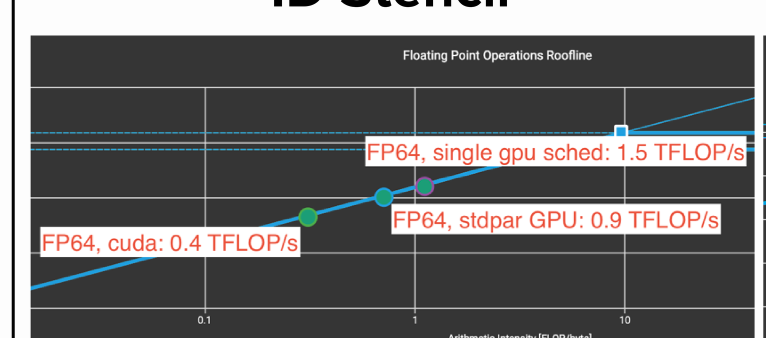
Speedups



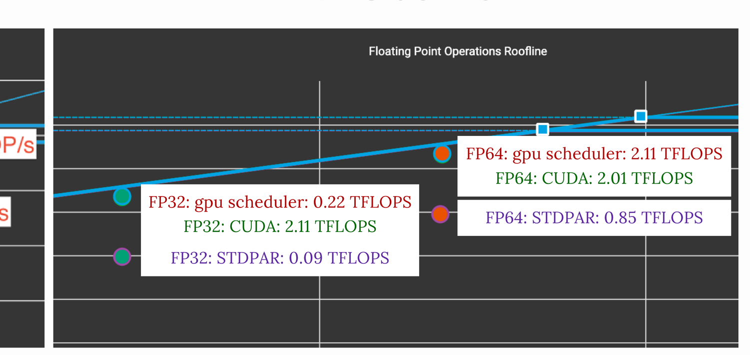


Roofline Analysis

1D Stencil

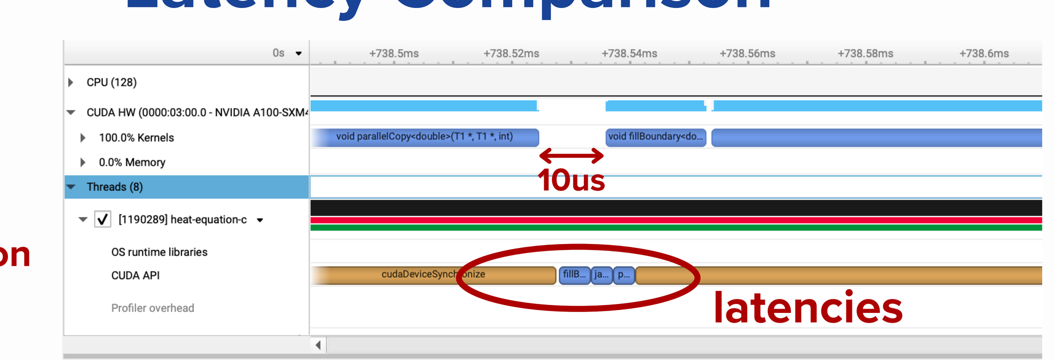


2D Stencil

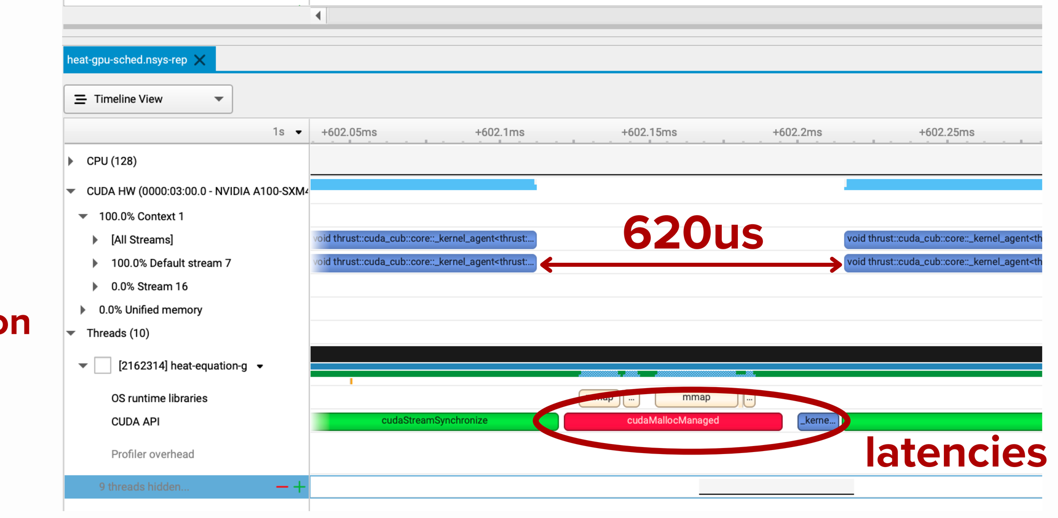


Latency Comparison

CUDA Implementation



stdexec Implementation



Discussion

Advantages

Performance: Virtually no overheads from ***stdexec***, possible to consolidate launch latencies.

Productivity: Same code runs on multiple CPUs and GPUs - simply swap **schedulers (stdexec)** or **compiler flags (stdpar)**

Portability: Standard C++ code can be compiled with other compilers for various architectures.

Generalizability: Architecture-independent model to program asynchronous task and data parallel codes.

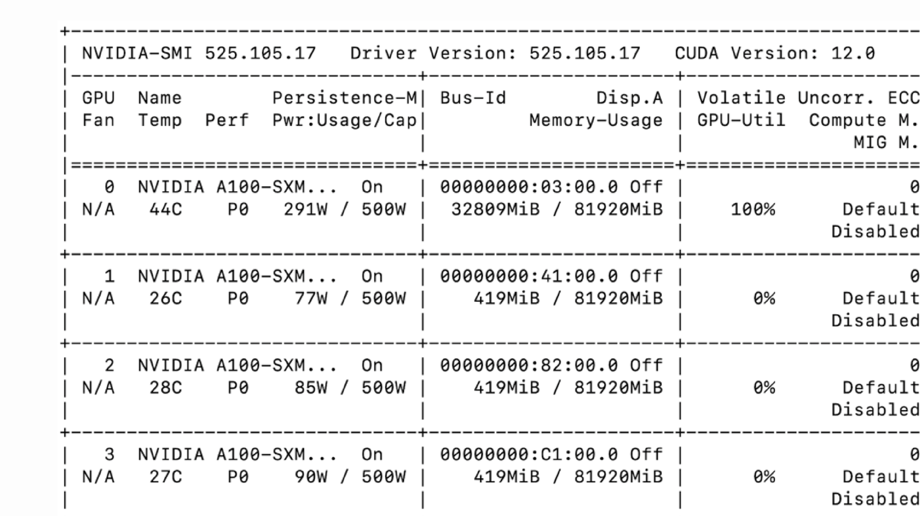
Limitations

Optimizations: Hardware or language specific optimizations are unavailable at user-level.

Parallelism Control: No explicit control over parallelism - number of blocks, threads or grid size.

Initial Setup: Learning curve to correctly use and set up language features, compilers, libraries, and dependencies.

Load Imbalance: MultiGPU and/or Multinode schedulers may introduce severe load imbalance depending on their implementations.



References

our code repo

stdexec proposal

nvidia stdexec

C++26 ADEPT