NERSC Users Group - NUG 2022

Amanda Sabatini Dufek
Lawrence Berkeley National Laboratory, NERSC, Berkeley, California, USA

Can we have a portable, performant software infrastructure that doesn't make application programmers rewrite their codes every few years, and what will it take to get there?

We hopefully can, but it will depend on the joint effort among application developers, compiler writers and hardware vendors.
One thing for sure: writing code in proprietary languages doesn't make sense anymore due to the increasing diversity of processor architectures.
Even if we lose some performance portability, I believe it is still worth it.

Code portability

OpenCL   low-level portable heterogeneous parallel programming model
SYCL   C++ single-source portable heterogeneous parallel programming model
OpenMP directive-based application programming interface

- ➔ portable code across architectures, vendors and generations; will always run
- ➔ support multiple heterogeneous devices (CPUs, GPUs, FPGAs,…)
- ➔ open-source, managed by the Khronos Group Inc

Performance portability

what to do rather than **how to do it**

what to do

declarative language rather than **procedural language**

how to do it

compilers + supported libraries

Dependency analysis

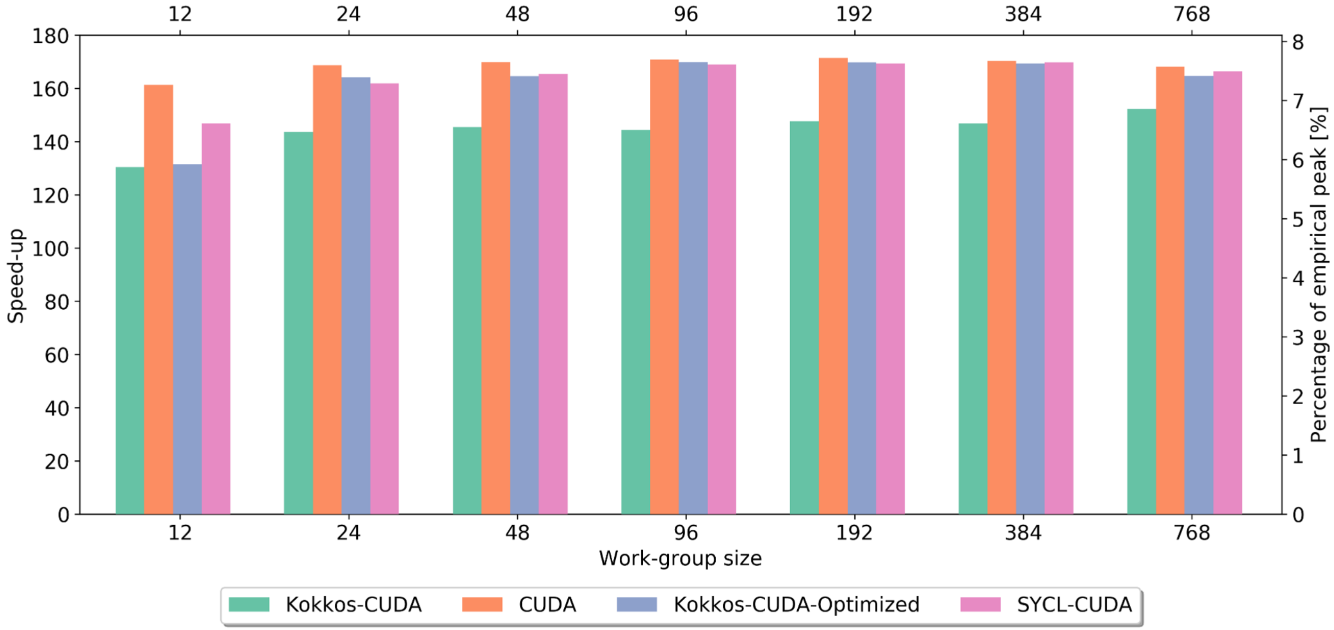
```
#include <CL/sycl.hpp>
#include <array>

int main() {

  std::array<int,N> a, b;
  for (int i=0; i<N; i++) { a[i] = b[i] = 0; }

  sycl::queue Q{gpu_selector{}};

  sycl::buffer A{a}, B{b};

  Q.submit([&](handler &h) {
      sycl::accessor accA(A, h, read_only);
      sycl::accessor accB(B, h, write_only);
      h.parallel_for(N, [=] (id<1> i) { accB[i] = accA[i] + 1; });
  });

  Q.submit([&](handler &h) {
      sycl::accessor accA(A, h, write_only);
      h.parallel_for(N, [=] (id<1> i) {  accA[i] = 42; });
  });

  Q.submit([&](handler &h) {
      sycl::accessor accB(B, h, write_only);
      h.parallel_for(N, [=] (id<1> i) { accB[i] = 52; });
  });

  Q.wait();

  return 0;
}
```

Reduction

```
#pragma omp target
{
  error = 0.0;
  #pragma omp target teams distribute parallel for reduction(max:error) collapse(2)
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);
      error = fmax( error, fabs(Anew[j][i] - A[j][i]));
    }
  }
}
```

Performance portability

The performance of different languages in different architectures depends on the compiler, problem, developer skill, and time effort.



Speed-up of the parallel Milc-Dslash kernel — implemented in CUDA, Kokkos, and SYCL, the last two with CUDA backend — as a function of work-group size on a single NVIDIA A100 GPU.

Paper: https://ieeexplore.ieee.org/document/9652859

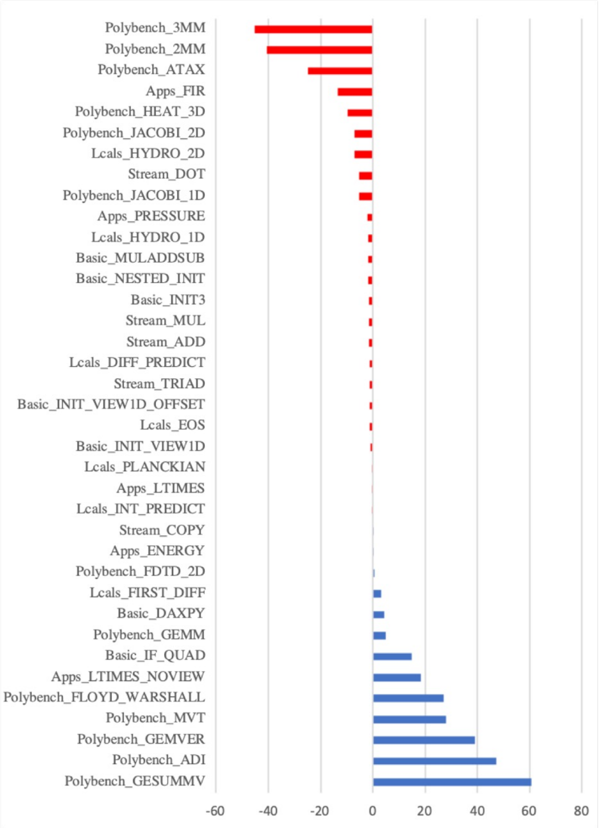


Figure 2: Percent speedup of SYCL kernels relative to CUDA kernels

Paper: https://dl.acm.org/doi/abs/10.1145/3388333.3388660