Evaluating ISO C++ Parallel Algorithms on Heterogeneous HPC Systems

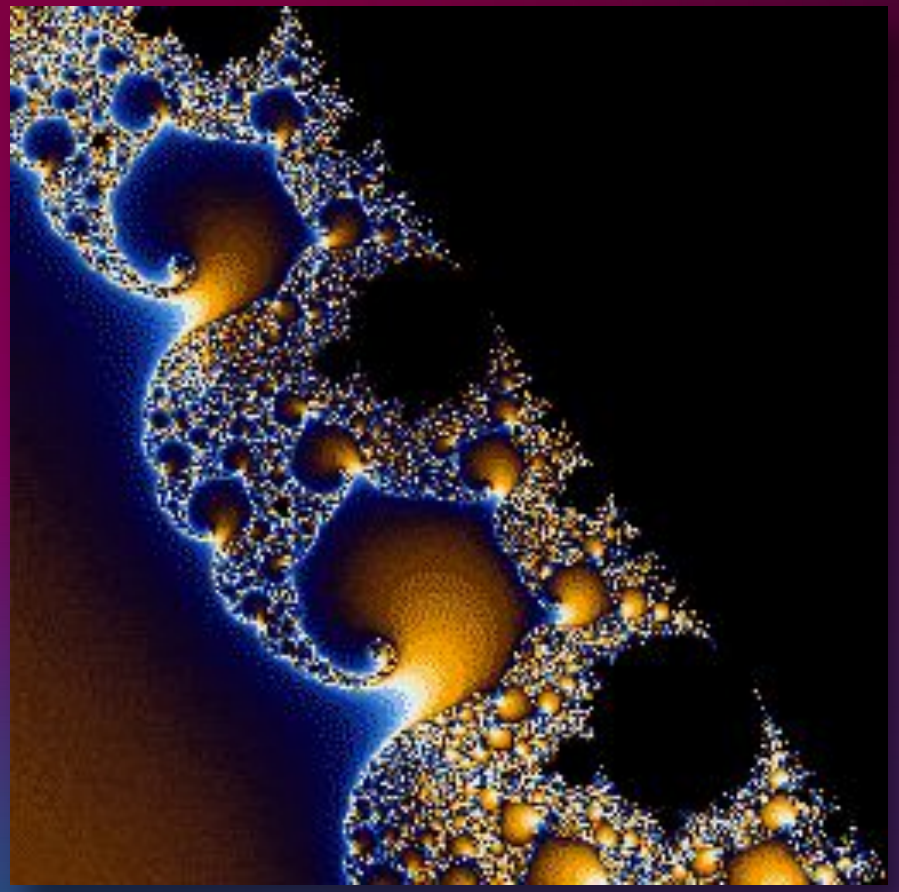
Wei-Chen (Tom) Lin, Tom Deakin, Simon McIntosh-Smith

Department of Computer Science

University of Bristol, UK

http://uob-hpc.github.io/

Motivating example: Mandelbrot



Animated mandelbrot GIF in < 60 lines of ISO C++ 17*
(*excludes GIF library, shim header)

- Code compiles as-is on Clang, GCC, DPC++, NVC++
  - Runs multithreaded on CPUs
  - Runs on NVIDIA and Intel GPUs

https://github.com/UoB-HPC/stdpar-mandelbrot

University of BRISTOL

Motivating example: Mandelbrot

```
struct Colour { uint8_t r, g, b, _ = 0xFF;  /* mix ... */ }

template <typename N>
std::pair<std::complex<N>, int> mandelbrot(std::complex<N> c, int imax) {
  std::complex<N> z; int i = 0;
  while (std::abs(z) <= 4 && i < imax) { z = z * z + c; i += 1; }
  return {z, i};
}

Colour *buffer = /* allocate W*H*Colour */
shim::ranged<size_t> r(0, width * height);
std::for_each(shim::par_unseq, r.begin(), r.end(), [=, buffer](auto i) {
  std::array<Colour, 16> Colours{Colour{66, 30, 15}, {25, 7, 26}, {9, 1, 47}, {4, 4, 73}};
  const auto &[t, iter] = mandelbrot(std::complex<double>{X, Y}, 360);
  if (iter < 360) { // smoothed escape time colouring
    auto logZn = std::log(std::abs(t)) / 2;
    auto nu = std::log(logZn / std::log(2)) / std::log(2);
    auto c1 = Colours[size_t(std::floor(iter - nu)) % Colours.size()];
    auto c2 = Colours[size_t(std::floor(iter + 1 - nu)) % Colours.size()];
    buffer[i] = c2.mix(c1, std::fmod(double(iter) + 1 - nu, 1));
  } else buffer[i] = {0, 0, 0};
});
```

Mandelbrot kernel excerpt

Modern C++ constructs

- General control structure
  - Function calls
  - while/for/if/else
- Structured bindings
- Lambda captures
- Pointers (if captured by value)
- POD types
- Stack allocation

https://github.com/UoB-HPC/stdpar-mandelbrot

University of BRISTOL

Overview

1. Background: ISO C++
2. C++ STL numeric/algorithm API
3. Index vs. Data traversal
4. C++17 PSTL Implementations
5. Experiments
   a. Benchmark Platforms
   b. Benchmark Mini-apps
   c. Evaluation Results
6. Conclusion

University of BRISTOL

Background: ISO C++

- Statically typed, unmanaged, low cost abstractions
- Rich standard library
- Committee driven: WG21
  - 3 year release cadence since C++11:
    - *C++14, C++17, C++20, C++23, …*
- Multiple compiler implementations
  - GCC from FSF
  - Clang from multiple vendors
    - DPC++/ICPX from Intel
    - AOCC from AMD
  - NVHPC from Nvidia
  - MSVC from Microsoft

Introduction: C++ STL numeric/algorithm API (Serial)

```
template<class InputIt,
         class UnaryF>
UnaryF for_each(InputIt inFirst, InputIt inLast, UnaryF f) {
    for (; inFirst != inLast; ++inFirst) f(*inFirst);
    return f;
}

template<class InputIt,
         class OutputIt,
         class UnaryF>
OutputIt transform(InputIt inFirst, InputIt inLast, OutputIt outFirst, UnaryF f) {
    while (inFirst != inLast) *outFirst++ = f(*inFirst++);
    return outFirst;
}

template<class InputIt,
         class T,
         class CombineF,
         class MapF>
T transform_reduce(InputIt inFirst, InputIt inLast, T t, CombineF reduce, MapF transform) {
    T acc = t;
    while (inFirst != inLast) acc = reduce(acc, transform(*inFirst++));
    return acc;
}
```

Introduction: C++ STL numeric/algorithm API (C++17)

```
template<class Policy, class InputIt,
         class UnaryF>
UnaryF for_each(Policy&& p, InputIt inFirst, InputIt inLast, UnaryF f) {

    // Offload implementation, selected with the p argument
}

template<class Policy, class InputIt,
         class OutputIt,
         class UnaryF>
OutputIt transform(Policy&& p, InputIt inFirst, InputIt inLast, OutputIt outFirst, UnaryF f) {

    // Offload implementation, selected with the p argument
}

template<class Policy, class InputIt,
         class T,
         class CombineF,
         class MapF>
T transform_reduce(Policy&& p, InputIt inFirst, InputIt inLast, T t, CombineF reduce, MapF transform) {


    // Offload implementation, selected with the p argument
}
```

- `std::execution::seq` - Ordered sequential execution
- `std::execution::unseq` - Unordered sequential execution
- `std::execution::par` - Parallel ordered execution
- `std::execution::par_unseq` - Parallel unordered execution

University of BRISTOL

Introduction: Data vs. Index Centric Traversal

Data Centric Traversal

```
auto exec = std::execution::par_unseq;

std::vector<T> xs = /*...*/;

std::for_each(exec, xs.begin(), xs.end(), [](T &x){
  /* ... */
});

std::vector<T> ys(xs.size());

std::transform(exec, xs.begin(), xs.end(), ys.begin(), [](T &x){
  return /* new value, witnessing x */
});
```

Index Centric Traversal (Naive)

```
auto exec = std::execution::par_unseq;

std::vector<T> xs = /*...*/;
// generate indices
std::vector<int> idxs(xs.size());
std::iota(idxs.begin(), idxs.end(), 0);
// idxs == {0,1,..xs.size()}
std::for_each(exec, idxs.begin(), idxs.end(), [](int
i){
  // i == 0 .. xs.size() - 1
});
```

University of BRISTOL

Introduction: Counting Iterator

```
template <typename N> struct range {
  struct iterator {
  friend class range;
  using difference_type = typename std::make_signed_t<N>;
  using iterator_category = std::random_access_iterator_tag;
  using value_type = N;
  using reference = N;
  using pointer = const N*;
  iterator &operator ++() { ++i_; return *this; }
  iterator operator ++(int) { iterator copy(*this); ++i_; return copy;
}
  // operator implementation for [],*,+,-,--,+=,-=,==,>=,<=,>,< omitted
  protected: explicit iterator(N start) : i_ (start) {}
  private: N i_;
  };
  iterator begin() const { return begin_; }
  iterator end() const { return end_; }
  range(N begin, N end) : begin_(begin), end_(end) {}
  private: iterator begin_, end_;
};
```

C++17

```
range<int> r(0, N);
std::for_each(r.begin(), r.end(), [](int i) {
 // i == 0 .. N
});
```

C++17

```
auto r = std::views::iota(0).begin();
std::for_each_n(r, N, [] (int i) {
  // i == 0 .. N
});
```

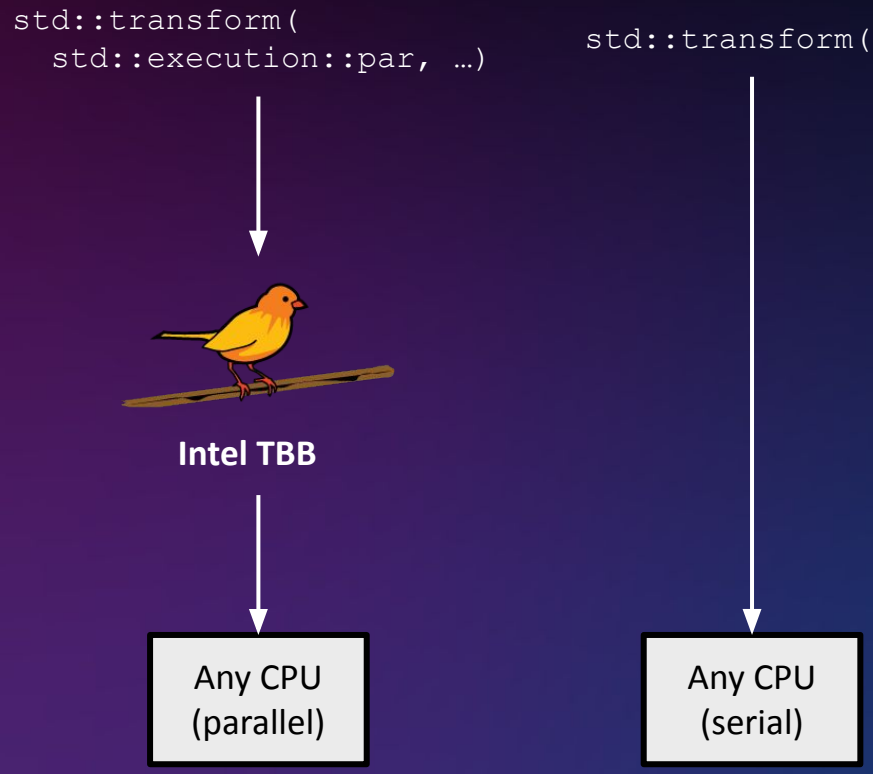
C++20

Introduction: ISO C++ PSTL Implementations

*Deprecated

C++17 Implementation: libstdc++ PSTL

- **Library**, GNU's C++ standard library impl.
  - Ships with GCC
- FOSS, GPLv3
- CPU only
  - C++17 parallel execution policy via Intel TBB
  - Implementation contributed by Intel



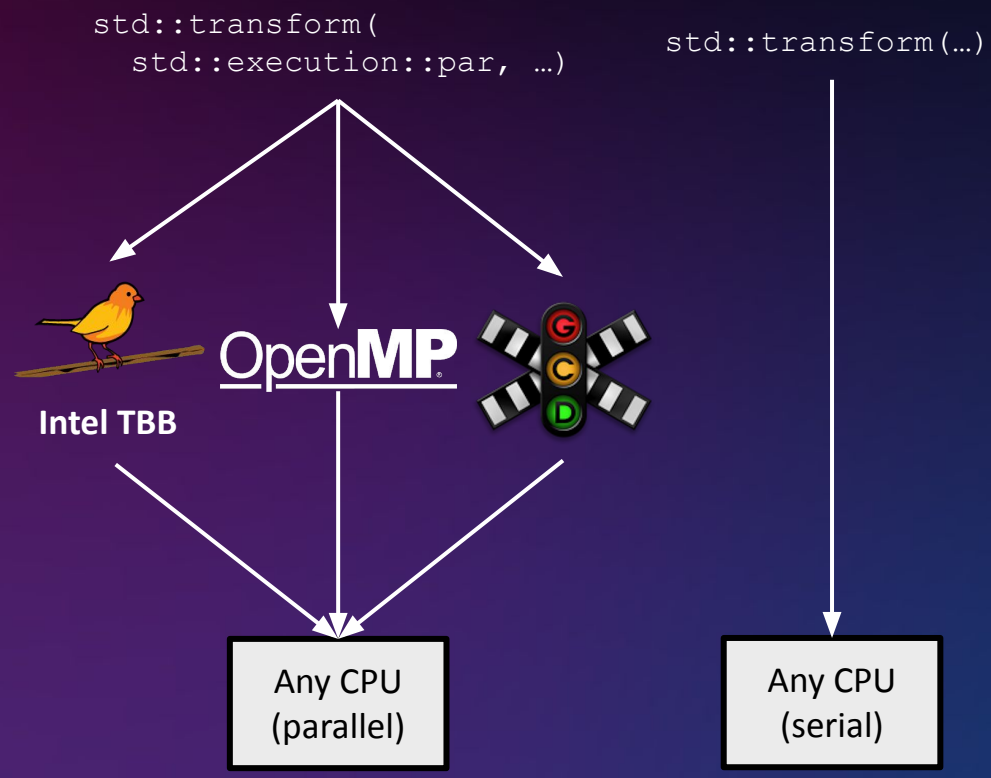
Intel Threading Building Block (TBB/oneTBB)

➢ General purpose concurrency primitive library

University of BRISTOL

C++17 Implementation: LLVM PSTL

- **Library**, part of the LLVM project
- Planned integration with libc++, LLVM's C++ std library
  - Less complete than libstdc++
  - Not frequently shipped with Clang
- FOSS, UIUC or Apache 2.0 w/ LLVM exception
- CPU only, multiple C++17 parallel execution policy backends
  - Intel TBB (Intel contribution)
  - OpenMP taskloops
  - macOS GCD
    - OpenMP not supported in AppleClang

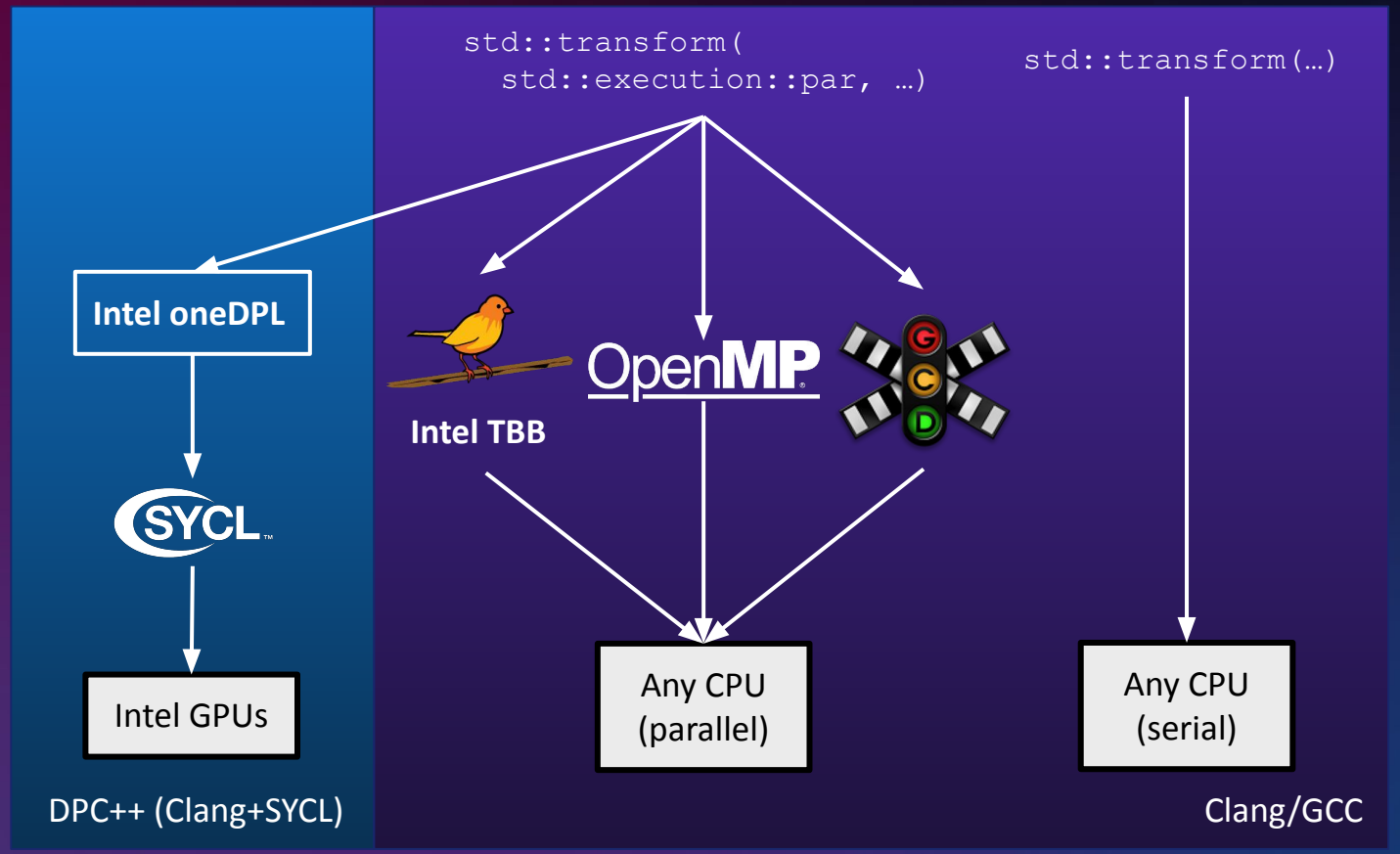


C++17 Implementation: Intel oneDPL

- Fork of the LLVM PSTL **library**
- FOSS, Apache 2.0 w/ LLVM exception
- CPU support untouched
- GPU offload implemented in **SYCL2020**
  - Kernels must adhere to SYCL2020 constraints
  - Validated on DPC++ **compiler**

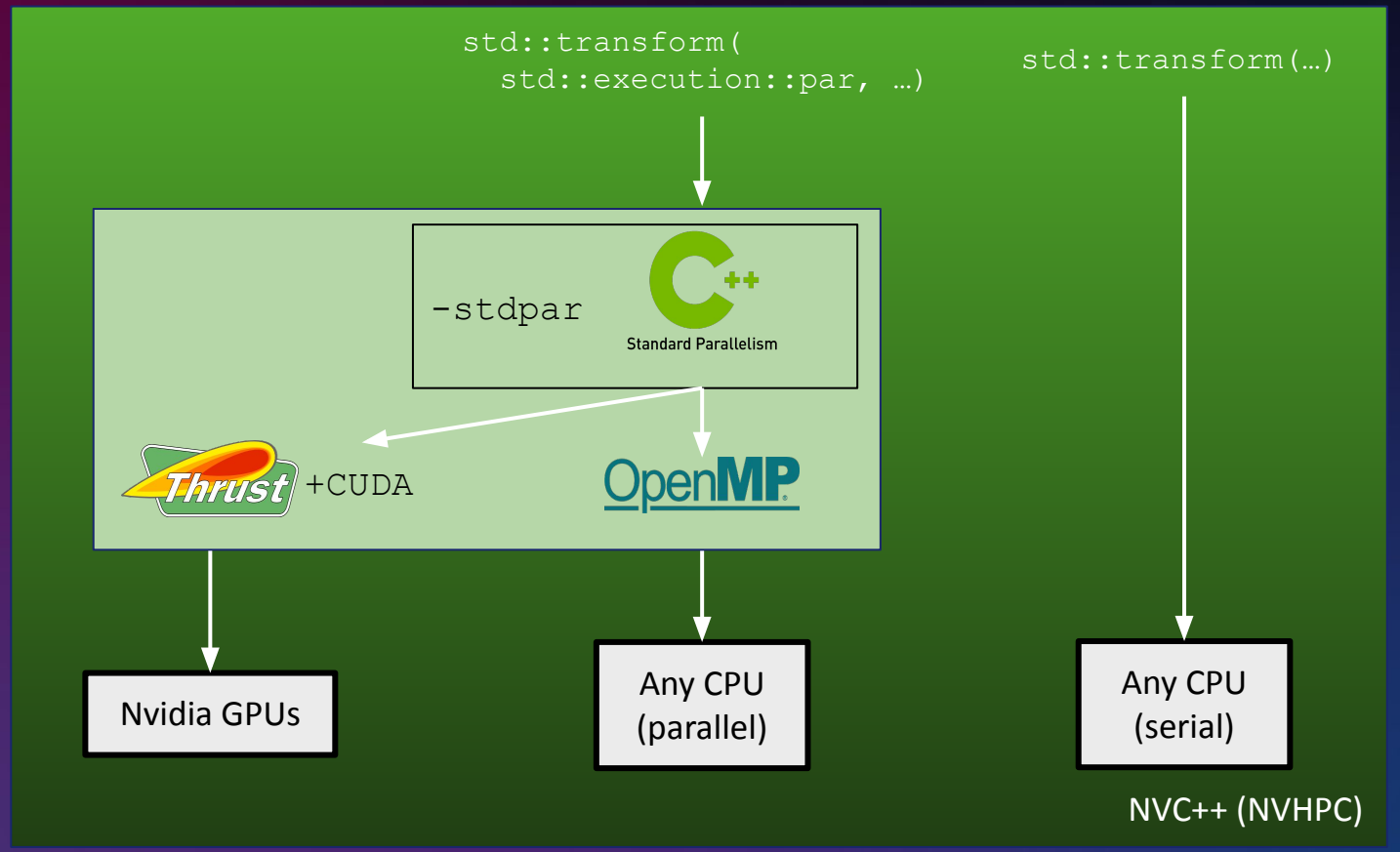
SYCL2020

- Single source accelerator dialect of C++
- Spiritual successor of OpenCL
- 2020 revision adds essential features
  - USM
  - Reduction



C++17 Implementation: NVIDIA NVHPC

- NVIDIA's heterogeneous **compiler**
- Closed source
- Formerly PGI Compilers
- Fully integrated SDK, compiler handles everything transparently



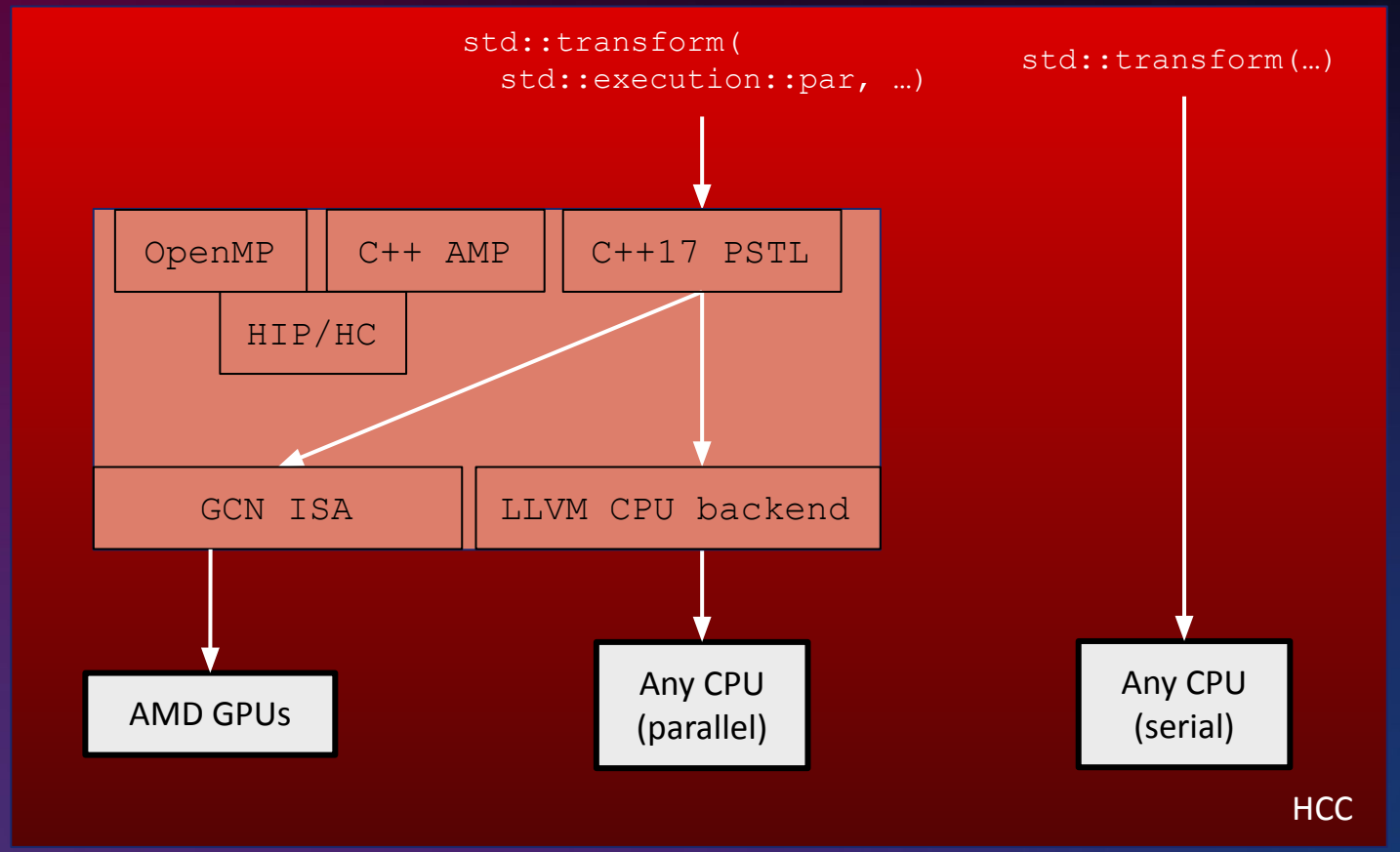
NVIDIA Thrust + CUDA

- CUDA: NVIDIA's accelerator C++ dialect
- Thrust: Productivity abstraction layer that implements STL
  - Multiple backends, including CUDA

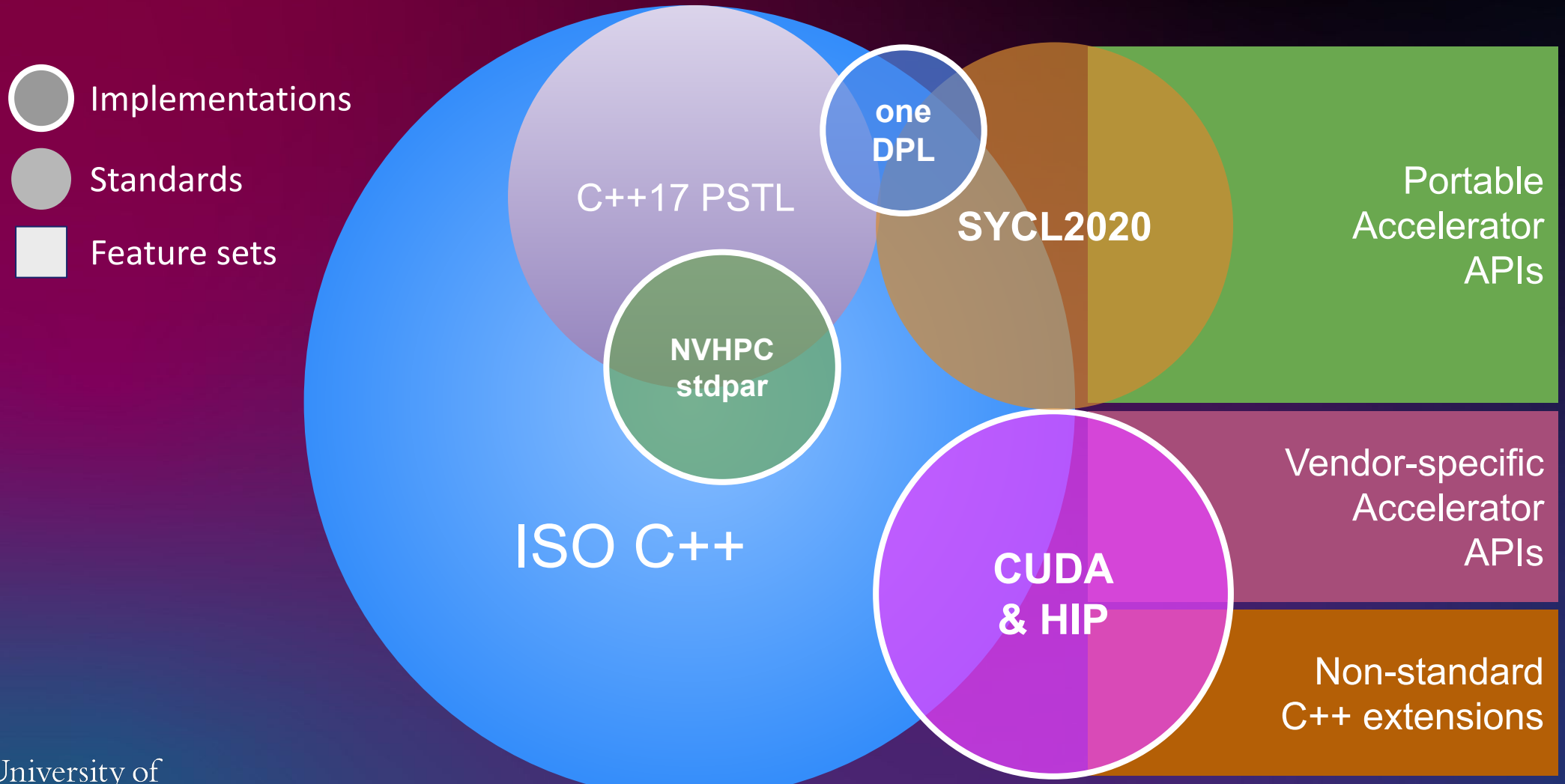
University of BRISTOL

C++17 Implementation: AMD HCC

- AMD's heterogeneous **compiler**
- FOSS, UIUC License
- **Deprecated**, unmaintained since 2019
  - Focus shifted to HIP and feature parity with NVIDIA's software stack
- Fully integrated SDK, compiler handles everything transparently



Expressing Parallelism



University of BRISTOL

Shim Header

```
#ifdef USE_ONEDPL // oneDPL C++17 PSTL
  #include <oneapi/dpl/algorithm>
  #include <oneapi/dpl/execution>
  #if ONEDPL_USE_DPCPP_BACKEND
    #include <CL/sycl.hpp>
namespace shim {
const auto par_unseq = oneapi::dpl::execution::device_policy<>{oneapi::dpl::execution::make_device_policy(cl::sycl::default_selector{})};
template <typename T> T *alloc_raw(size_t size) { return sycl::malloc_shared<T>(size, par_unseq.queue()); }
template <typename T> void dealloc_raw(T *ptr) { sycl::free(ptr, par_unseq.queue()); }
} // namespace shim
  #else
namespace shim {
static constexpr auto par_unseq = dpl::execution::par_unseq;
}
    #define USE_STD_PTR_ALLOC_DEALLOC
  #endif
#else // Normal C++17 PSTL
  #include <algorithm>
  #include <execution>
namespace shim {
static constexpr auto par_unseq = std::execution::par_unseq;
}
  #define USE_STD_PTR_ALLOC_DEALLOC
#endif
#ifdef USE_STD_PTR_ALLOC_DEALLOC
namespace shim {
template <typename T> T *alloc_raw(size_t size) { return reinterpret_cast<T *>(std::malloc(sizeof(T) * size)); }
template <typename T> void dealloc_raw(T *ptr) { std::free(ptr); }
} // namespace shim
#endif
```

Experiments

Port representative HPC mini-apps to C++17 and compare performance

- Existing mini-app must already have implementation in multiple models
- Characteristics
  - Memory-bandwidth bound
  - Compute bound
  - Complex, multi-kernel application
- Comprehensive platform
  - x86 and AArch64 CPUs
  - All supported GPUs

University of BRISTOL

Experiments

AArch64 and x86 CPUs
- Kokkos (OpenMP backend, Clang+GCC+NVHPC)
- OpenMP (Clang+GCC+NVHPC)
- C++17 (NVHPC)
- C++17 (TBB backend; Clang+GCC)
- C++17 (oneDPL OpenMP backend; Clang+GCC)

Nvidia GPUs
- Kokkos (CUDA backend, NVHPC)
- OpenMP Target (NVHPC)
- CUDA (NVHPC)
- C++17 (NVHPC)

Intel GPUs
- Kokkos (SYCL backend, DPC++)
- OpenMP Target (ICPX)
- SYCL2020 (DPC++)
- C++17 (oneDPL SYCL; DPC++)

Vendor	Name	Architecture	Abbreviation	Device Type	Total NUMA nodes
Intel	Xeon Gold 6338	x86, Ice Lake	Xeon	HPC CPU (32C*2)	2 (1 per socket)
AMD	EPYC 7713	x86, Zen3 (Milan)	EPYC	HPC CPU (64C*2)	8 (4 per socket)
AWS	Graviton 2	AArch64, Neoverse N1	Graviton2	HPC CPU (64C*1)	1
AWS	Graviton 3	AArch64, Neoverse V1	Graviton3	HPC CPU (64C*1)	1
NVIDIA	Tesla A100 (SXM 40GB)	Ampere	A100	HPC GPU	N/A
NVIDIA	Tesla V100 (PCIe 16GB)	Volta	V100	HPC GPU	N/A
Intel	UHD P630 (Xeon E2176G)	Gen9.5	UHD	Server iGPU	N/A
Intel	IrisPro 580 (i7 6670HQ)	Gen9	IrisPro	Consumer iGPU	N/A

Mini-app: BabelStream

- Memory-bandwidth bound
- Source code available on GitHub
  - https://github.com/UoB-HPC/BabelStream
- Port of the McCalpin STREAM benchmark to paradigms
  - Abstraction libraries:
    - **Kokkos**, RAJA
  - C/C++ dialects
    - **SYCL**, OpenCL, **CUDA**, HIP
  - C/C++ directives
    - **OpenMP**, OpenMP target, OpenACC
  - Libraries
    - **Intel TBB**, NVIDIA Thrust,
  - Languages
    - **ISO C++17 (data&index)**, Rust, Julia, Scala, Java

Algorithm 1 BabelStream kernels

1: **procedure** COPY($A[n], C[n], n$)
2: **for** $i \leftarrow 0$ **to** n **do** $C[i] \leftarrow A[i]$
3: **procedure** MUL($A[n], B[n], C[n], scalar, n$)
4: **for** $i \leftarrow 0$ **to** n **do** $B[i] \leftarrow scalar * C[i]$
5: **procedure** ADD($A[n], B[n], C[n], n$)
6: **for** $i \leftarrow 0$ **to** n **do** $C[i] \leftarrow A[i] + B[i]$
7: **procedure** TRIAD($A[n], B[n], C[n], scalar, n$)
8: **for** $i \leftarrow 0$ **to** n **do** $A[i] \leftarrow B[i] + (scalar * C[i])$
9: **procedure** DOT($A[n], B[n], scalar, n$)
10: **for** $i \leftarrow 0$ **to** n **do** $sum \leftarrow sum + (A[i] * B[i])$
return sum

University of BRISTOL

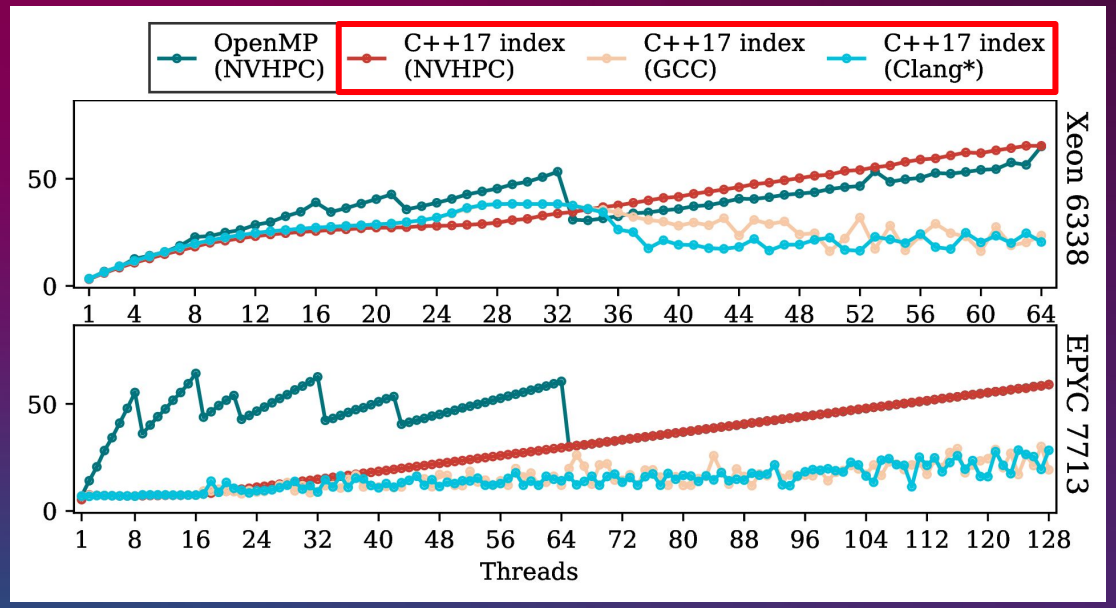
Porting: BabelStream

```
// OpenMP
#pragma omp parallel for
for (int i = 0; i < array_size; i++)
  c[i] = a[i] + b[i];


// Data centric
std::transform(exe_policy, a, a + array_size, b, c, [](auto l, auto r){
  return l + r;
});

// Index centric
ranged<int> range(0, array_size);
std::transform(exe_policy, range.begin(), range.end(), c, [a, b](int i)
{
  return a[i] + b[i];
});
```

University of BRISTOL

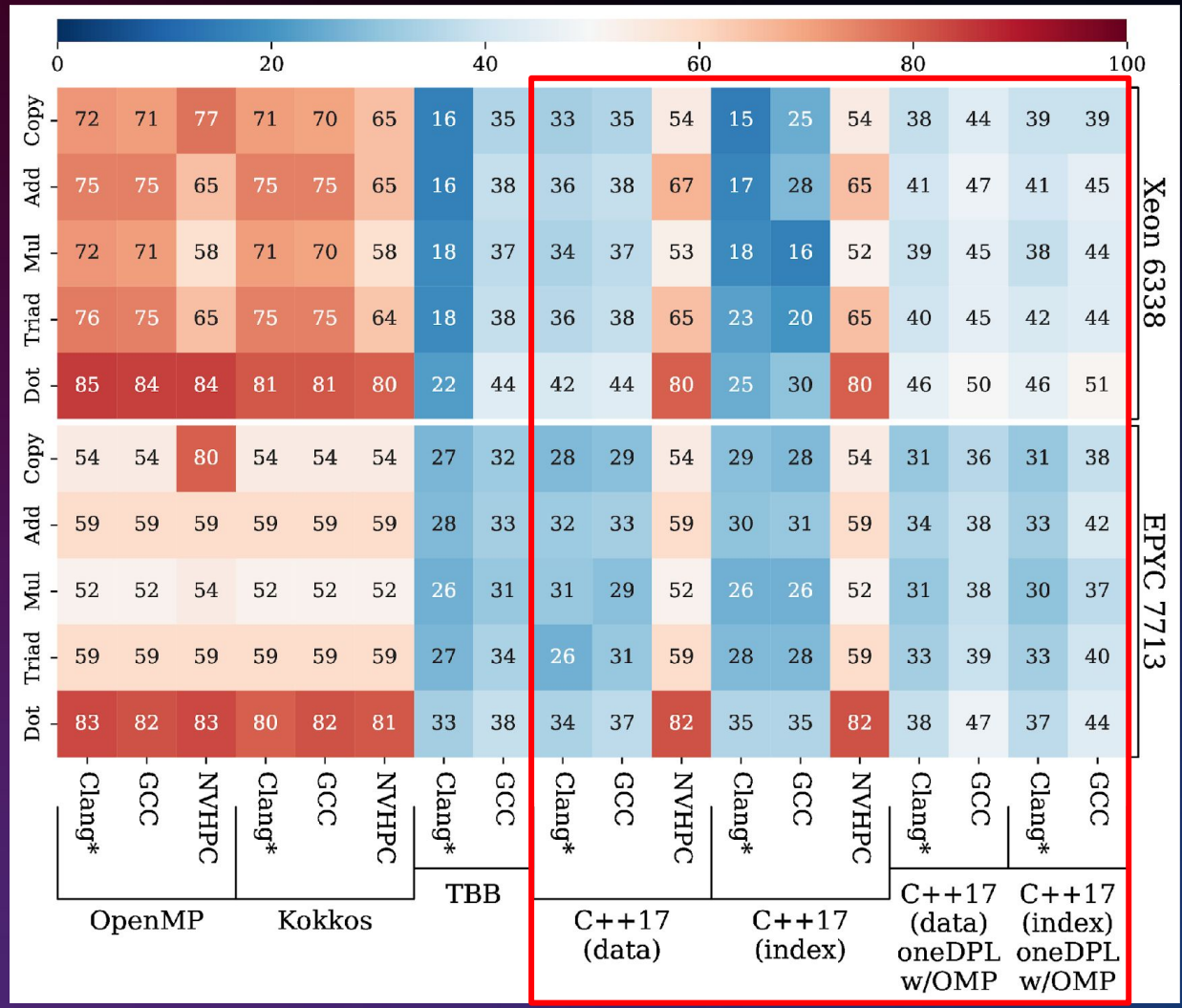
Results: BabelStream x86 CPU

- Explicit NUMA-awareness
  - OpenMP, C++17 on OpenMP
- Bad chucking + OpenMP taskloop
  - oneDPL



CPU thread scaling

% of peak memory bandwidth per platform; higher is better

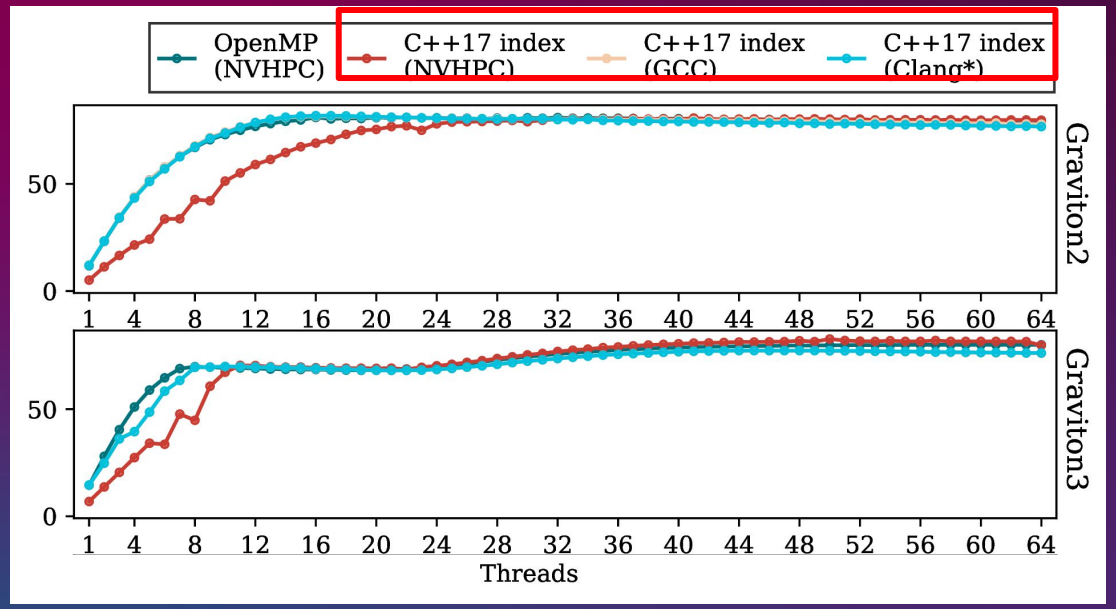


Relative bandwidth

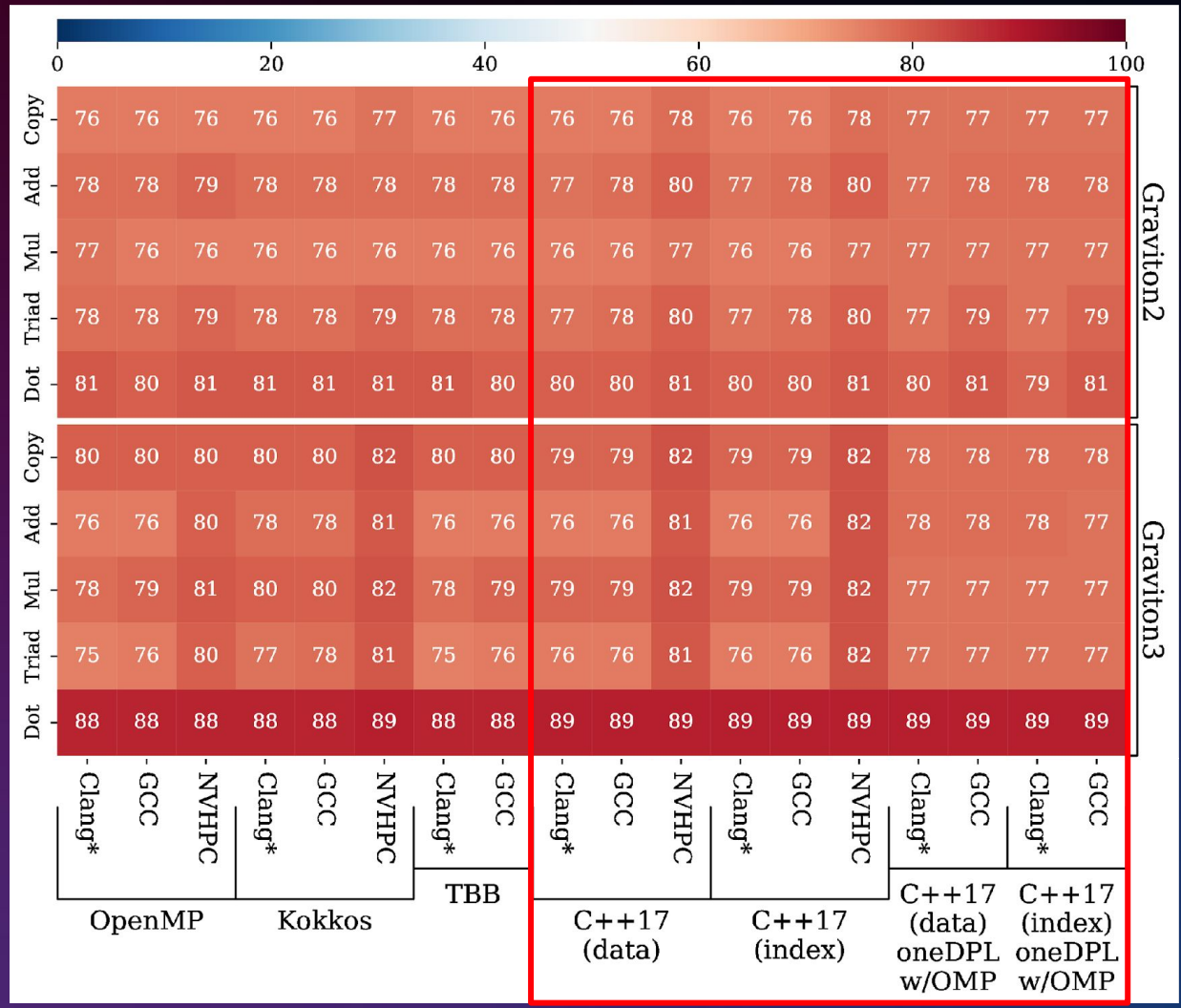
Results: BabelStream AArch64 CPU

% of peak memory bandwidth per platform; higher is better

- Graviton nodes lack NUMA regions
- Good performance overall, mature compilers



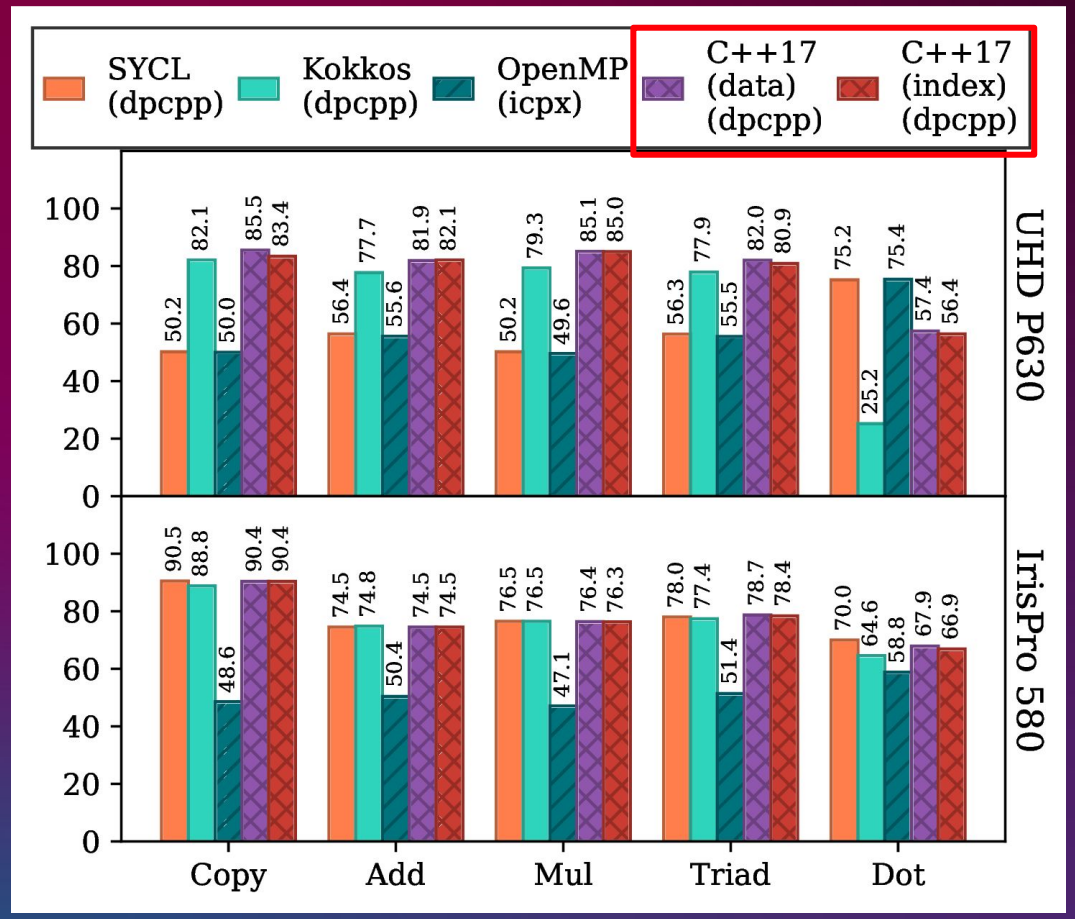
CPU thread scaling



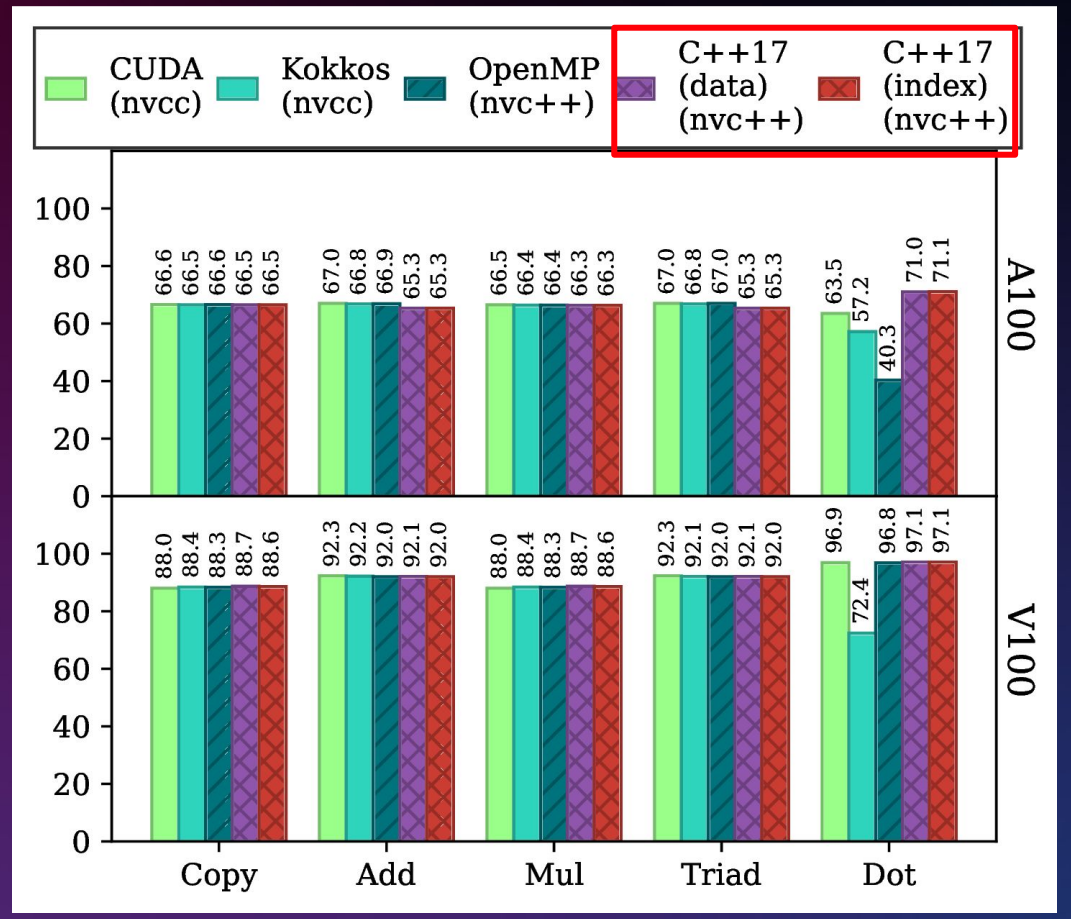
Relative bandwidth

Results: BabelStream GPUs

% of peak memory bandwidth per platform; higher is better



Intel GPUs

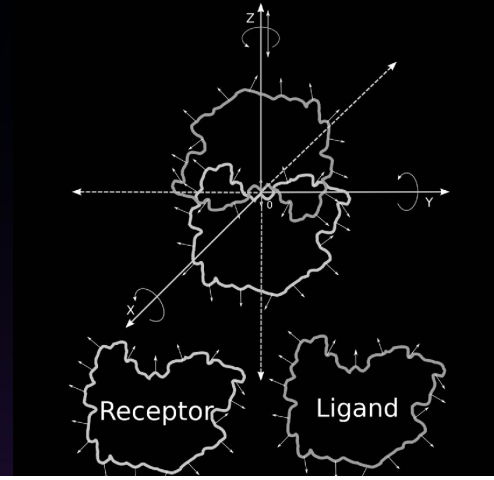


NVIDIA GPUs

University of BRISTOL

Mini-app: miniBUDE

- Proxy application of the Bristol University Docking Engine (BUDE)
- Source code available on GitHub
  - https://github.com/UoB-HPC/miniBUDE
- Compute bound, measurements in GFLOP/s or total runtime
  - Abstraction libraries:
    - **Kokkos**, RAJA
  - C/C++ dialects
    - **SYCL**, OpenCL, **CUDA**, HIP
  - C/C++ directives
    - **OpenMP**, OpenMP target, OpenACC
  - Libraries
    - **Intel TBB**, NVIDIA Thrust,
  - Languages
    - **ISO C++17 (Index)**, Julia



Algorithm 2 miniBUDE Fasten Kernel

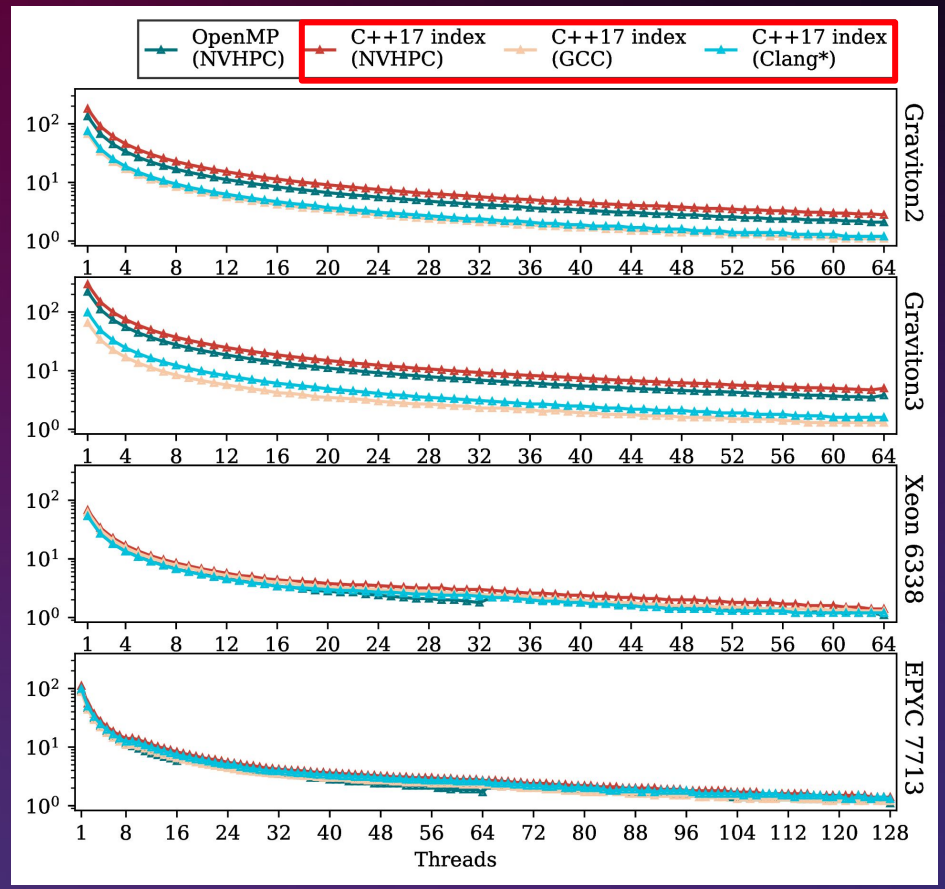
1: **procedure** FASTEN(const i, const $xform_{3\times3}[]$, const $proteins[ps]$, const $ligands[ls]$, out $energy[]$)
▷ Values $R, DSLV, DSLV_R, NZ, DST_1, DST, HRD, T$ are part of the simulation constants
2: **for** $il \leftarrow 0, ls$ **do**
3: $lpos_{1\times3} \leftarrow xform \cdot ligands[il].pos_{1\times3}$
4: **for** $ip \leftarrow 0, ps$ **do**
▷ Atom distance and sphere radii sum
5: $dist \leftarrow distance(lpos, proteins[ip].pos_{1\times3})$
6: $d \leftarrow dist - R$
▷ Steric energy, formal/dipole charge interactions
7: $energy[i] \leftarrow energy[i] +$ $(1 - dist * (1/R)) * (d < 0?2 * HRD : 0)$
8: $e \leftarrow init *$ $(d < 0.f?1 : (1 - d * DST_1)) *$ $(d < DST?1 : 0)$
9: $energy[i] \leftarrow energy[i] + (typeE? - |e| : e) * T$
▷ Nonpolar-Polar repulsive interactions
10: $dslvE = dslvInit *$ $((d < DSLV \wedge NZ)?1 : 0.f) *$ $(d < 0?1 : (1 - d * DSLV_R))$
11: $energy[i] \leftarrow energy[i] + dslvE * 0.5$

University of BRISTOL

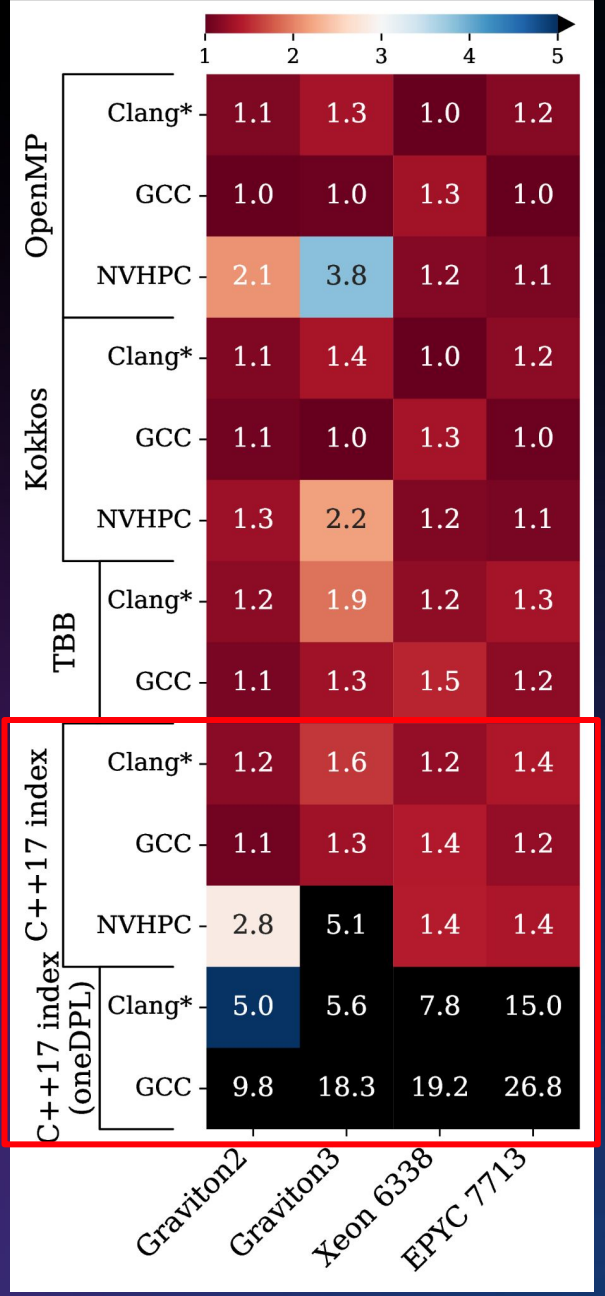
Results: miniBUDE CPUs

- Unaffected by NUMA
- NVHPC unoptimised for ARM
  - ARM support in NVHPC is new
- Same chunking+OpenMP taskloop issue on oneDPL

Results as normalised runtime per platform; lower is better



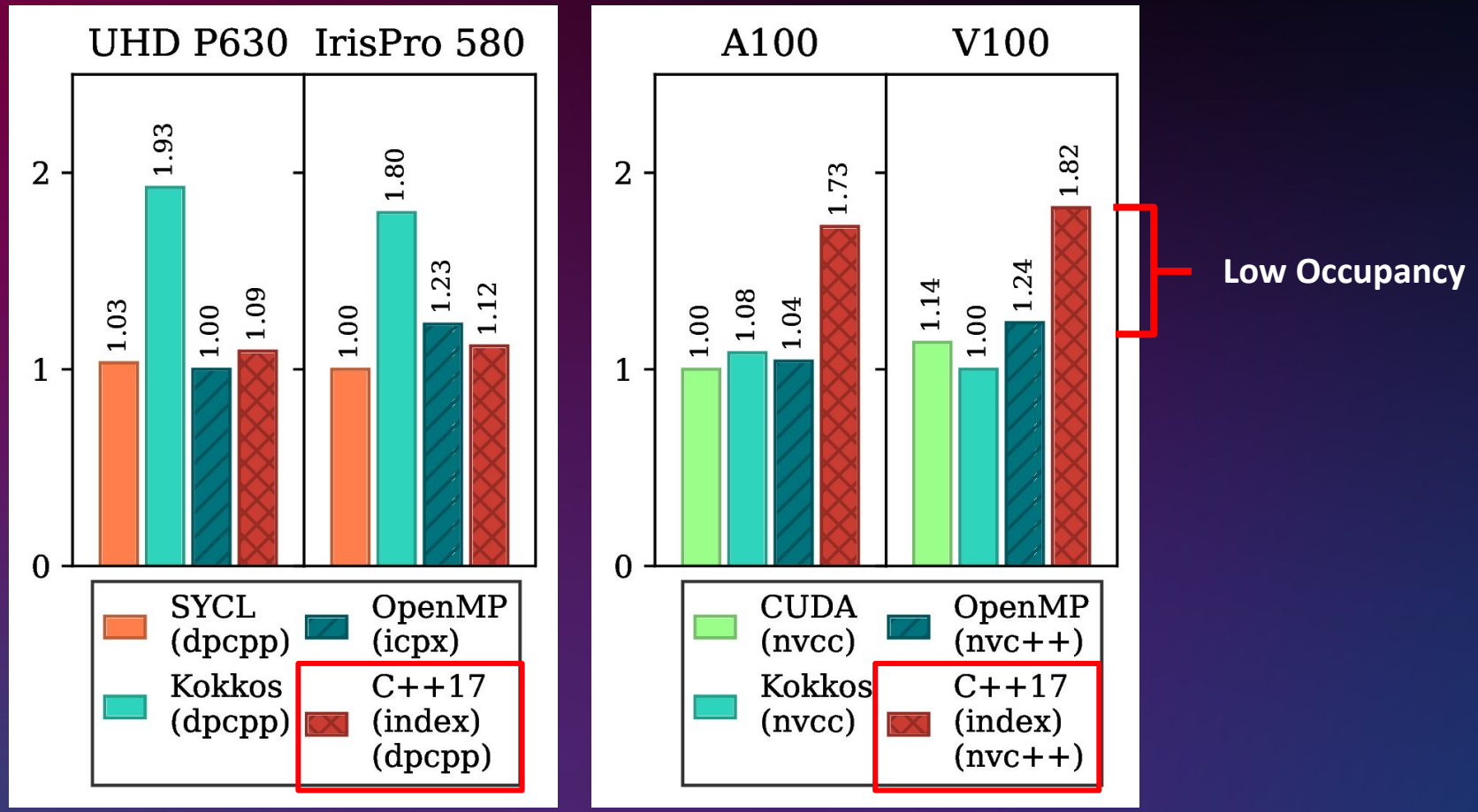
CPU thread scaling



Relative runtime

Results: miniBUDE GPUs

Results as normalised runtime per platform;lower is better



Intel GPUs

NVIDIA GPUs

Mini-app: CloverLeaf

- Proxy application for 2D hydrodynamics
- Source code available on GitHub
  - https://github.com/UoB-HPC/cloverleaf_stdpar
- Mixed memory-bandwidth bound; measurements in total runtime
  - Structured grid; stencil access pattern
  - Reductions
- Complex application, >100 unique kernels + MPI halo exchange
  - Abstraction libraries:
    - **Kokkos**
  - C/C++ dialects
    - **SYCL**, OpenCL, **CUDA**
  - C/C++ directives
    - **OpenMP**, OpenMP target
  - Libraries
    - **Intel TBB:**
  - Languages
    - **ISO C++17 (Index)**:

Algorithm 3 High-level CloverLeaf kernel overview

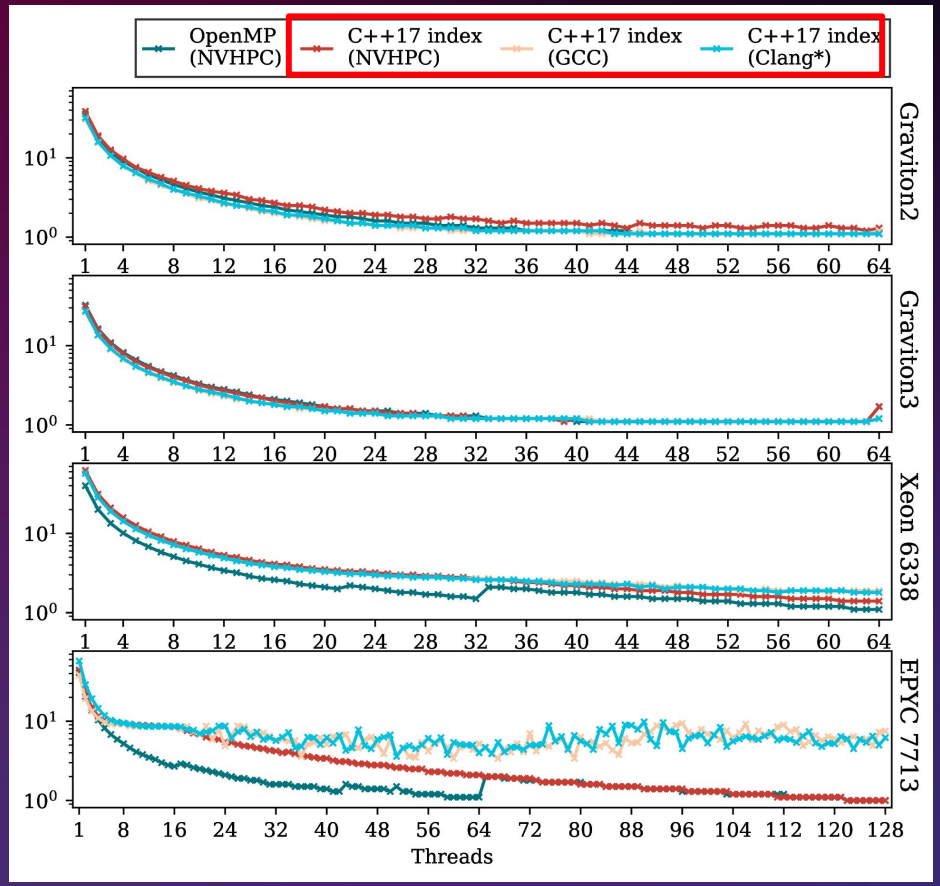
▷ Each procedure traverses the full $W \times H$ grid, some procedures may invoke multiple kernels
1: **while** $step < maxStep$ **do**
2: **procedure** IDEAL_GAS
▷ Pressure/sound speed via ideal gas equation of state with a fixed gamma
3: **procedure** VISCOSITY
▷ Artificial viscosity via the Wilkin's method to smooth out shock front and prevent oscillations
4: **procedure** PDV
▷ Cell energy/density δ via velocity gradients
5: **procedure** CALC_DT
▷ Compute the minimum timestep based on CFL conditions, velocity gradient, and velocity divergence.
6: **procedure** ACCELERATE
▷ Update velocity field via cell pressure/viscosity gradients
7: **procedure** FLUX_CALC
▷ Edge volume fluxes using the velocity fields
8: **procedure** ADVECTION
▷ Setup fields for the next iteration
9: **procedure** RESET_FIELD
▷ Edge volume fluxes based on the velocity fields
10: **procedure** FIELD_SUMMARY
▷ Total mass, internal energy, kinetic energy, and volume weighted pressure
11: $step \leftarrow step + 1$

University of BRISTOL

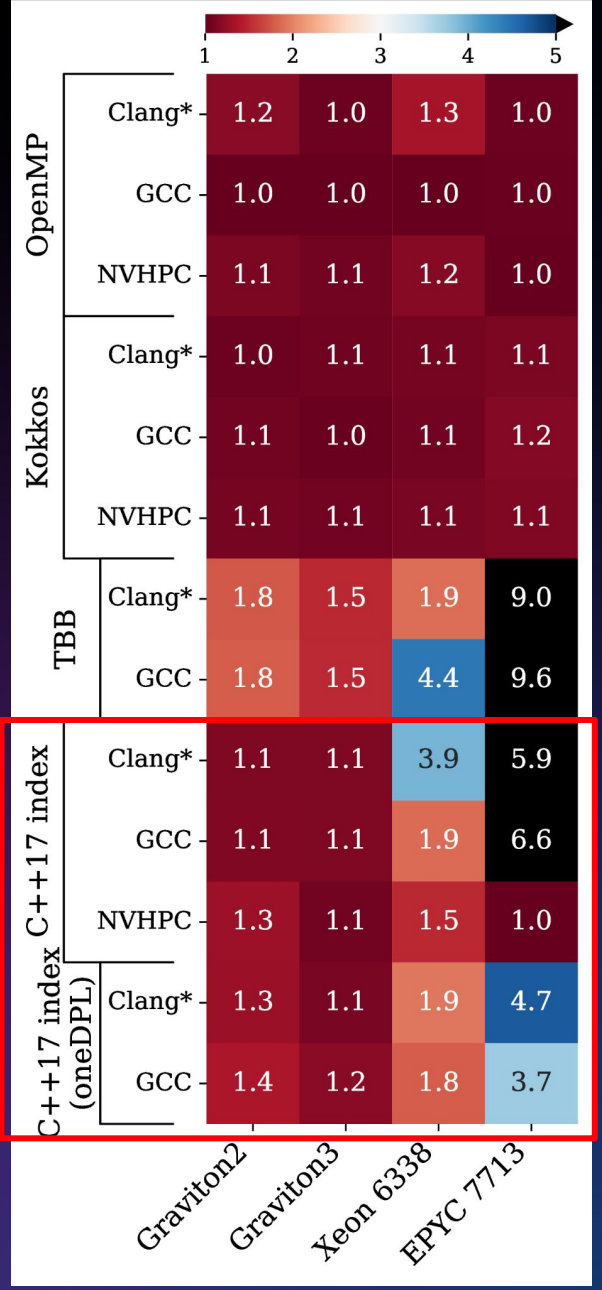
Results: CloverLeaf CPUs

- Strong NUMA effects
  - Similar to BabelStream
- Degradation proportional to NUMA domain count

Results as normalised runtime per platform; lower is better



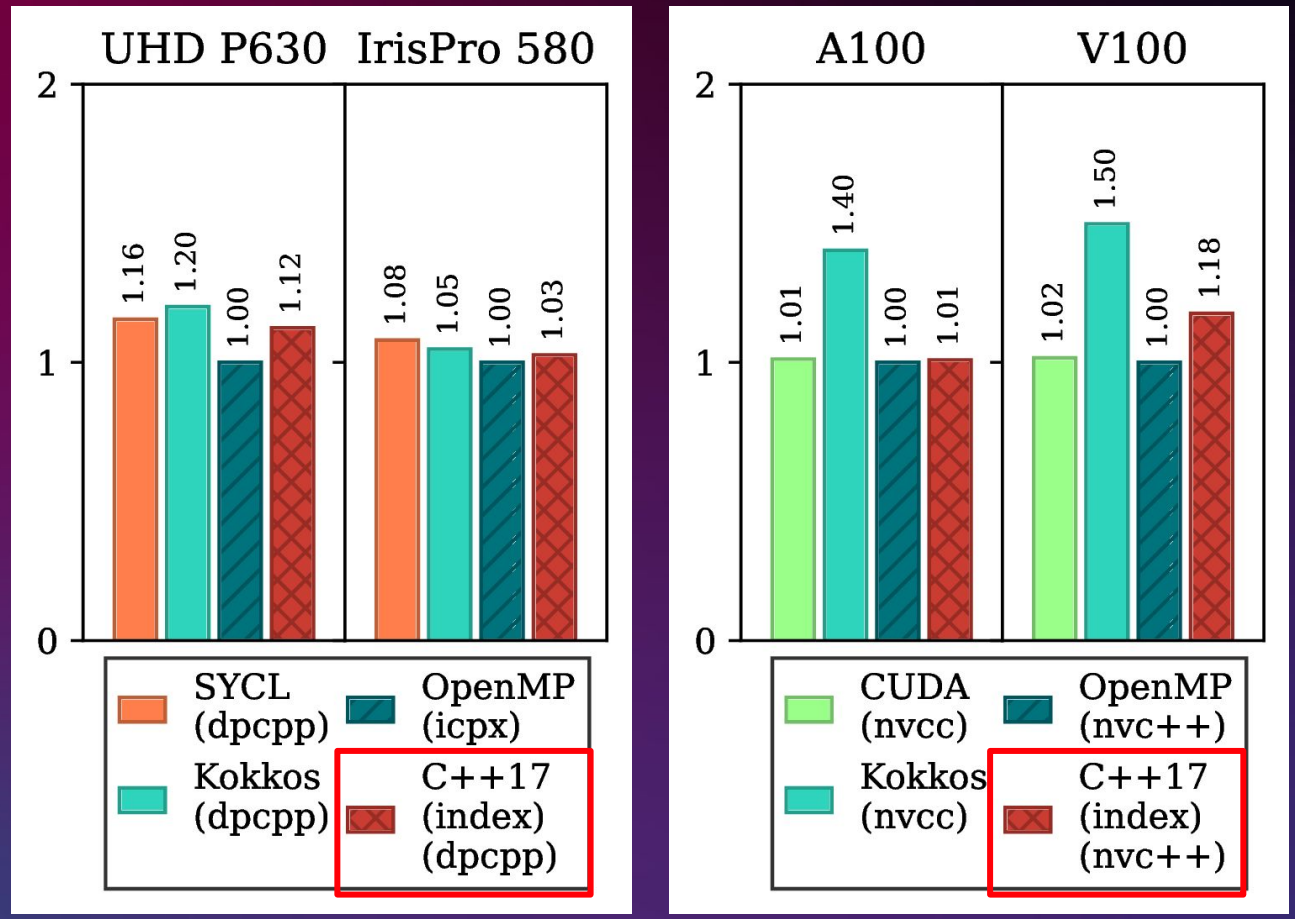
CPU thread scaling



Relative runtime

Results: CloverLeaf GPUs

Results as normalised runtime per platform;lower is better

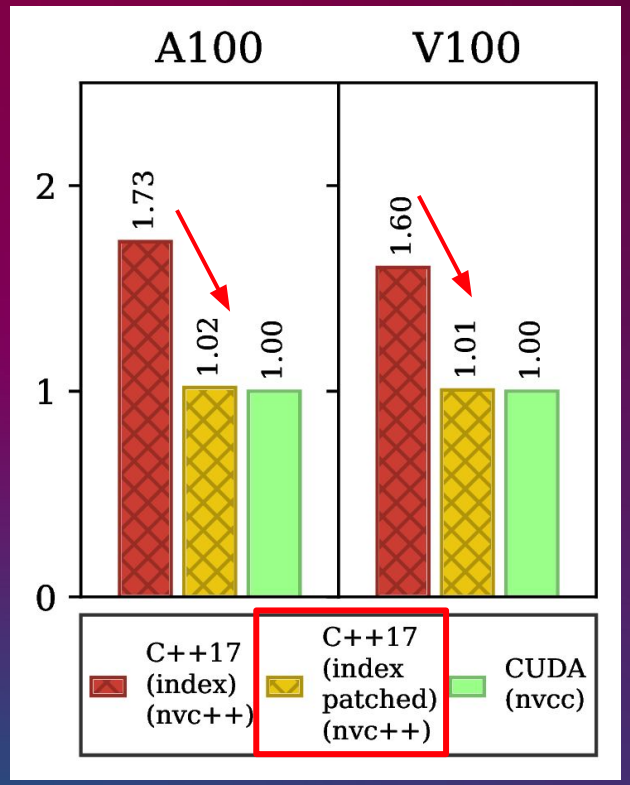


Intel GPUs

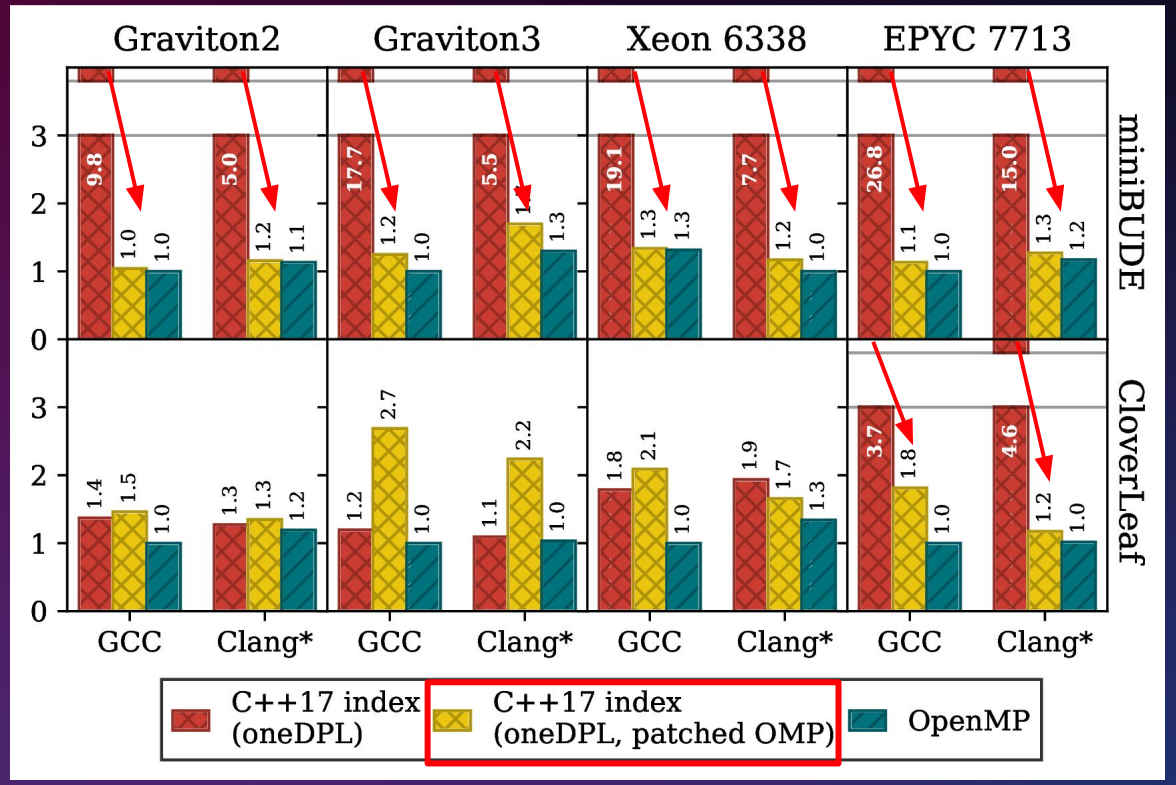
NVIDIA GPUs

Performance issues

miniBUDE Thrust header patch uplift



oneDPL OpenMP chunk patch uplift



Conclusion

- Three ISO C++ ports of representative mini-apps
  - In high productivity, portable, idiomatic C++
    - Reductions
    - Transformations
  - Traditional index-central traversal works
    - Stencil
    - Multiple allocations in kernel
- Performance portable
  - Directly comparable to established models in most cases
- Missing pieces
  - Device control - no API
  - Async dispatch - senders/receivers coming soon
  - >= C++20 features - Coming soon to NVHPC and GCC/Clang

University of BRISTOL

References

University of Bristol HPC Group
http://uob-hpc.github.io

[IJCSE] *Evaluating attainable memory bandwidth of parallel programming models via BabelStream*
Deakin T, Price J, Martineau M, McIntosh-Smith S.
https://dl.acm.org/doi/10.5555/3292750.3292751 DOI: 10.1504/IJCSE.2018.095847

[ISC21] *A Performance Analysis of Modern Parallel Programming Models Using a Compute-Bound Application*
Andrei Poenaru, Wei-Chen Lin and Simon McIntosh-Smith
https://link.springer.com/chapter/10.1007/978-3-030-78713-4_18 DOI: 10.1007/978-3-030-78713-4_18

[PMBS21] *Comparing Julia to Performance Portable Parallel Programming Models for HPC*
Wei-Chen Lin and Simon McIntosh-Smith
https://ieeexplore.ieee.org/document/9652798 DOI: 10.1109/PMBS54543.2021.00016

University of BRISTOL