# Poster submission: Modern C++ for Parallelism in High Performance Computing

**Victor Eijkhout**

**CppCon 2024**

## Introduction

This poster reports on 'D2D', a benchmark that explores elegance of expression and performance in the context of a High Performance Computing 'mini-application'. The same code has been implemented using a number of different approaches to parallelism. Implementations are discussed with performance results.

## Relevance

C++ is making inroads into HPC / Scientific Computing, a field traditionally dominated by C and Fortran. With all the developments in modern C++ such as range algorithms, their parallel execution strategies, and multi-dimensional arrays through 'mdspan', it is interesting to explore what C++ can offer for lower level performance critical operations.

Scientific computing is an interesting test cases since many algorithms are fairly regular and therefore should scale perfectly in parallel, with fairly predictable performance. The D2D benchmark explores to what extent we achieve scaling for different parallelization strategies: C-style programming with OpenMP, native mechanisms in modern C++, as well as through Kokkos and Sycl.

## Discussion

An important corner of scientific computing is concerned with sparse matrices, in particular so-called 'stencil' operations on large arrays, structured as two or three-dimensional grids. A stencil operation computes each location in an output array from a small number of points in an input array. In our benchmark we use the 2nd order Laplace operator:

$$\forall_{i,j} : y_{ij} \leftarrow 4x_{ij} - x_{i-1,j} - x_{i+1,j} - x_{i,j-1} - x_{i,j+1}$$

where $i, j$ can jointly range over up to billions of points.

This type of operation has two important characteristics:

1. Since all outputs are independent, it is perfectly parallelizable. (This is sometimes called ‘data parallelism’.) Therefore, achieved parallel speedup will be an important measured quantity in D2D.
2. One impediment to perfect parallel execution is that this code is bandwidth-limited, so we expect parallel efficiency to stop scaling at a certain core count. It is an interesting question whether some parallelism models have other limitations that make them scale worse than others.

*Computational structure* Our mini-app is the ‘power method’: an algorithm for computing the largest eigenvalue of a matrix, but which with modifications can stand for Google’s PageRank, as well as a large class of numerical analysis algorithms.

It comprises the following operations:

- An array scaling, which is a perfectly parallel operation.
- A norm computation. This is a ‘reduction’, which would be done with an atomic update in a general threading model, but is usually implemented more efficiently with local accumulators. Hence we expect similar performance to the scaling.
- The stencil application has a more complicated structure, and loads more data. Exact analysis of the performance involves some hardware considerations.

Our interest lies in comparing the performance of different parallelization strategies. We are mostly interested in overall performance, but we will present some profiling to analyze the individual operations.

*Parallelism Models* In scientific computing, the two dominant parallelism models are MPI (Messsage Passing Interface) for distributed memory, and OpenMP (Open Multi-Processing) for shared memory. In this project we focus mostly on the shared memory aspect and use OpenMP as the performance baseline.

(1) OpenMP has standard bindings to C and Fortran, so we start with a plain ‘C-style’ implementation based on simple loops and linear vectors for floating point data storage. (2) OpenMP also has bindings to C++, where it can exploit a random access iterator. This means that we reimplement the algorithm with range-based loops.

(3) Using more modern mechanisms we explore using ‘mdspan’ for true two-dimensional indexing. We use range algorithms in the sense that we range over a cartesian product of iota views, giving us the two-dimensional index into the mdspan.

Next we explore two extensions to a pure C++ approach, both strongly based on the use of lambda expressions.

(4) Kokkos is a C++ library that facilitates our type of data parallelism. It originated as an execution layer of the massive Trilinos scientific library, but has gained popularity independent of that. We explore whether the generality of a library layer has performance impact.

(5) The second extension is Sycl, a Khronos open standard, which can be considered a library built on standard C++, but in practice requires compiler support. The major proponent of Sycl is Intel, which provides a compiler in its latest release. The alternative is AdaptiveCPP, a cross compiler based on Clang, and targeting various parallelism backends, including OpenMP.

## Completion status

At present the implementations described above are completed and have been tested on various CPUs. We present relevant code snippets illustrating the parallelization strategies, with graphs and tables of results. Code profiling and discussion of performance.

*By presentation time:* So far, exclusively top-of-the-line high core count processors were used. I will compare to a more ordinary desktop, where bandwidth limitations will be much more severe.

Most models are capable of running – with varying amounts of effort – on GPUs. The software / hardware integration this requires makes it less than trivial, but I will have some results.

The parallel implementation using execution policies met with various problems. I hope to have time to investigate this.

*Future work:* The 'AdaptiveCPP' open source alternative to Sycl is very hard to install. I hope to find time for another attempt. This will allow testing Sycl on two different compilers: Intel with optimized TBB backends, and AdaptiveCPP with a more common backend.

## Supporting material

The benchmark is in an open repository: https://github.com/VictorEijkhout/diff2d_benchmark

## Bio and contact info

Victor Eijkhout, Texas Advanced Computing Center, 10100 Burnet Rd, Austin TX 78758, eijkhout@tacc.utexas.edu

Victor Eijkhout is a research scientist at the University of Texas at Austin. His interests include numerical algorithms and parallel computing. He teaches courses in C++ and scientific/parallel computing. He is the author of the popular 'The Art of HPC' book series.