Extending and Reproducing pSTL-Bench: oneDPL Backend Integration and Enhanced Reproducibility

Oleg Bilovus
Department of Computer Science, University of Salerno
Fisciano (SA), Italy

Abstract

Describe in concise words what you do, why you do it (not necessarily in this order), and the main result. The abstract has to be self-contained and readable for a person in the general area. You should write the abstract last.

1 Introduction

Writing performance-portable and efficient parallel applications remains a significant challenge due to the diversity of modern hardware architectures. The emergence of parallel programming frameworks and, more recently, the introduction of parallel algorithms in the *C++17 Standard Template Library (STL)* aim to address this challenge by enabling developers to write code that runs efficiently across CPUs and GPUs using a standardized interface.

pSTL-Bench [1] is a benchmark suite originally developed to quantitatively evaluate the performance of individual parallel STL algorithms across various compiler frameworks (such as GCC, Intel's icpx, NVIDIA HPC SDK) and backends (including TBB, HPX, OpenMP, and CUDA). It provides a focused, micro-benchmarking approach that avoids the complexities of full applications and highlights the relative performance characteristics of different STL implementations.

This Work. In this work, the main results and experiments from the original pSTL-Bench paper are reproduced. Beyond reproduction, the project is then extended by incorporating support for the *oneAPI DPC++ Library (oneDPL)* backend, enabling the evaluation of Intel's parallel STL implementation based on the *SYCL programming model* on both CPU and GPU platforms

To further enhance the usability and reproducibility of the benchmark suite, automation tools are introduced, including Ansible playbooks for environment setup and benchmark execution, R scripts for performance analysis, and expanded documentation for ease of deployment and adding new backends.

Related Work. This work builds directly on pSTL-Bench. While performance portability and parallelism have been widely studied in the context of full applications [2] and frameworks such as Kokkos [3] and RAJA [4], no prior work has reproduced or extended pSTL-Bench, nor added support for the oneDPL backend or focused on improving its reproducibility.

2 Background

This section introduces the relevant concepts and context for evaluating the scalability of parallel algorithms implemented in the C++ Standard Template Library (STL). It defines the scope of the problem, outlines the algorithms under analysis, and describes the supporting frameworks and benchmarking strategies used for performance assessment.

Parallel STL in C++. Since C++17, the Standard Template Library (STL) includes parallel versions of many common algorithms, enabling hardware parallelism through familiar interfaces. Compiler support varies, relying on different runtimes and libraries to target platforms such as multi-core CPUs and GPUs, with differences in scheduling, memory use, and synchronization.

Problem Definition. The primary objective is to analyze how different implementations of parallel STL algorithms scale with increasing problem sizes and hardware resources. This includes identifying the computational overheads and the performance gains achieved through parallel execution. A detailed performance comparison is necessary to understand which combinations of compilers and backends provide the most efficient use of available computing resources.

Algorithms. The study focuses on five representative algorithms that embody diverse computational patterns:

- **find** – Searches for a target value within a range using linear traversal.
- **for_each** – Executes a loop with k_{it} increments per element to simulate computation and stores the result. k_{it} is a parameter that can be adjusted to control the workload.
- **inclusive_scan** – Produces a prefix sum, where each element in the output is the sum of all previous elements plus the current element, requiring synchronization.
- **reduce** – Computes a single result by summing all elements in a range, which can be parallelized but requires synchronization to combine results.
- **sort** – Rearranges elements into ascending order, requiring synchronization and data movement.

Backends. In this work, support for the *oneAPI DPC++ Library (oneDPL)* is added, which is based on the SYCL programming model and enables parallel STL execution on both CPUs and GPUs.

The original pSTL-Bench suite includes the following backends:

- **GNU Parallel STL** – The GCC implementation of parallel algorithms using OpenMP.
- **TBB** – A C++ library offering task-based parallel algorithms and data structures.
- **HPX** – A C++ runtime for fine-grained parallel and distributed applications.
- **OpenMP** – A standard API for shared-memory parallel programming.
- **CUDA** – NVIDIA's platform for general-purpose GPU computing.

The evaluation covers various compiler-backend combinations, including those based on GNU, Intel, and NVIDIA compilers.

Automation. To facilitate reproducibility and ease of use, the project includes automation tools:

- **Ansible** – An agentless automation tool that simplifies the setup of the benchmark environment, including installation

of dependencies, configuration of the system, compilation of the benchmark suite, and execution of tests.
- **Semaphore UI** – A web-based interface for managing and monitoring the execution of ansible playbooks, providing a user-friendly way to trigger tests and view results.
- **R Scripts** – Scripts for analyzing performance data, generating plots similar to those in the original paper. and providing insights into the performance characteristics of different implementations.

3 Benchmark Setup

This section describes the hardware and software setup used for the benchmarking experiments, including platform specifications, compiler versions, and the configuration of the parallel STL implementations.

Hardware Platform. The benchmarks were executed on a *DELL G16 7630* laptop connected to external power, with the hardware specifications shown in Table 1.

Table 1: Hardware configuration

CPU [5]	
Processor	Intel Core i9-13900HX
Cores	24 (8 P-cores + 16 E-cores)
Core Frequency	2.20 GHz (up to 5.40 GHz)
Threads	32 threads
Memory	32 GB RAM
GPU [6]	
GPU	NVIDIA GeForce RTX 4070 (mobile)
CUDA Cores	4608 CUDA cores
Core Frequency	1230 MHz (up to 2175 MHz)
Memory	8 GB GDDR6 VRAM

Software Platform. The benchmarks were executed on a Linux-based operating system, specifically *Ubuntu 25.04 Desktop*. The desktop version was used due to GPU driver compatibility issues with the server edition. Prior to execution, the CPU frequency governor was set to *performance* mode to ensure maximum processor performance [7].

Compilers and Libraries. The compilers and libraries used in the experiments are listed in Table 2 and Table 3, respectively.

Table 2: Compilers

Compiler	Version
g++	14.2.0
Intel icpx	2025.1.1
NVIDIA nvc++	25.3-0

Table 3: Libraries

Library	Version
oneDPL	2022.8
TBB	2022.1
HPX	1.9.1
NVOMP	25.3
CUDA	12.8

Benchmark Configuration. All benchmarks were configured the same as the original study, with one exception: the problem size in certain experiments was reduced from 2^{30} to 2^{29}. This change was necessary because the oneDPL backend crashed on the GPU when using the larger problem size. Listing 1 shows the error message produced during execution with the original size.

Listing 1: oneDPL GPU runtime error output

```
<CUDA>[ERROR]:
UR CUDA ERROR:
  Value:           2
  Name:            CUDA_ERROR_OUT_OF_MEMORY
  Description:     out of memory
  Function:        allocateMemObjOnDeviceIfNeeded
  Source Location: /builds/oneapi-core/oneapi-release-
       ↪ package/intel-llvm-mirror/build/_deps/unified-
       ↪ runtime-src/source/adapters/cuda/memory.cpp:442

<CUDA>[ERROR]:
UR CUDA ERROR:
  Value:           2
  Name:            CUDA_ERROR_OUT_OF_MEMORY
  Description:     out of memory
  Function:        allocateMemObjOnDeviceIfNeeded
  Source Location: /builds/oneapi-core/oneapi-release-
       ↪ package/intel-llvm-mirror/build/_deps/unified-
       ↪ runtime-src/source/adapters/cuda/memory.cpp:442

terminate called after throwing an instance of 'sycl::_V1
    ↪ ::exception'
  what():  Native API failed. Native API returns: 38 (
       ↪ UR_RESULT_ERROR_OUT_OF_HOST_MEMORY)
```

4 Experimental Results

Here you evaluate your work using experiments. You start again with a very short summary of the section. The typical structure follows.

Experimental setup. Specify the platform (processor, frequency, maybe OS, maybe cache sizes) as well as the compiler, version, and flags used. If your work is about performance, I strongly recommend that you play with optimization flags and consider also icc for additional potential speedup.

Then explain what kind of benchmarks you ran. The idea is to give enough information so the experiments are reproducible by somebody else on his or her code. For sorting you would talk about the input sizes. For a tool that performs NUMA optimization, you would specify the programs you ran.

Results. Next divide the experiments into classes, one paragraph for each. In each class of experiments you typically pursue one questions that then is answered by a suitable plot or plots. For example, first you may want to investigate the performance behavior with changing input size, then how your code compares to external benchmarks.

Comments:

- Create very readable, attractive plots (do 1 column, not 2 column plots for this report) with readable font size. However, the font size should also not be too large; typically it is smaller than the text font size. An example is in Fig. (of course you can have a different style).
- Every plot answers a question. You state this question and extract the answer from the plot in its discussion.
- Every plot should be referenced and discussed.

5 Conclusions

Here you need to summarize what you did and why this is important. *Do not take the abstract* and put it in the past tense. Remember, now the reader has (hopefully) read the report, so it is a very different

situation from the abstract. Try to highlight important results and say the things you really want to get across such as high-level statements (e.g., we believe that .... is the right approach to .... Even though we only considered x, the .... technique should be applicable ....) You can also formulate next steps if you want. Be brief. After the conclusions there are only the references.

References

[1] R. Laso, D. Krupitza, and S. Hunold, "Exploring Scalability in C++ Parallel STL Implementations" in *Proceedings of the 53rd International Conference on Parallel Processing (ICPP '24)*, Association for Computing Machinery, New York, NY, USA, 2024, pp. 284–293. https://doi.org/10.1145/3673038.3673065

[2] W. -C. Lin, T. Deakin and S. McIntosh-Smith, "Evaluating ISO C++ Parallel Algorithms on Heterogeneous HPC Systems" in *2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, Dallas, TX, USA, 2022, pp. 36-47. https://doi.org/10.1109/PMBS56514.2022.00009

[3] H. C. Edwards and C. R. Trott, "Kokkos: Enabling Performance Portability Across Manycore Architectures" *2013 Extreme Scaling Workshop (xsw 2013)*, Boulder, CO, USA, 2013, pp. 18-24. https://doi.org/10.1109/XSW.2013.7

[4] D. A. Beckingsale et al., "RAJA: Portable Performance for Large-Scale Scientific Applications" *2019 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, Denver, CO, USA, 2019, pp. 71-81. https://doi.org/10.1109/P3HPC49587.2019.00012

[5] Intel, "Intel Core i9-13900HX Processor" *Intel SKU*, https://www.intel.com/content/www/us/en/products/sku/232171/intel-core-i913900hx-processor-36m-cache-up-to-5-40-ghz/specifications.html

[6] NVIDIA, "GeForce RTX 4070 Laptop GPU" *GeForce RTX 40 Series Laptops*, https://www.nvidia.com/en-us/geforce/laptops/40-series/

[7] Google Benchmark, "Reducing Variance, Disabling CPU Frequency Scaling" *Google Benchmark Documentation*, https://google.github.io/benchmark/reducing_variance.html