

Exploring Scalability in C++ Parallel STL Implementations

International Conference on Parallel Processing 2024 — ICPP'24

Ruben Laso, Diego Krupitza, and Sascha Hunold

`{laso, krupitza, hunold}@par.tuwien.ac.at`

August 13, 2024

Research Group for Parallel Computing, TU Wien



Informatics

C++ in High-Performance Computing

- Languages like Fortran, C, and C++ are the most used in HPC
 - Existing code base
 - Performance
 - Compatibility with new hardware
- C++ can be used in several types of architectures
 - Multi-core and many-core CPUs → OpenMP, TBB
 - GPUs → CUDA, OpenCL
 - FPGAs → SYCL
- Should we use a different code for each system?

Performance Portability

Same code performs “well” on different architectures

Performance Portability in C++

- Several libraries: Kokkos, RAJA, HPX, ...
- **C++17** with **execution policies**
 - Parallel execution of the algorithms in STL (standard library)
 - Different compilers/backends

Questions

- **Speedup and efficiency** of parallel STL algorithms?
- Which is the **best compiler/backend**? GCC vs ICC, TBB vs OpenMP, ...
- **GPUs** performance?

Contributions

- *pSTL-Bench*: micro-benchmark suite
- Evaluation of the performance for
 - Different compilers: GCC, ICC, NVIDIA HPC SDK
 - Different backends: OpenMP, TBB, HPX, CUDA
 - Different systems: Intel and AMD CPUs, NVIDIA GPUs

pSTL-Bench

- Code available on github.com/parlab-tuwien/pSTL-Bench
- Suite of **micro-benchmarks** to test performance portability in C++
 - STL algorithms: `std::for_each`, `std::reduce`, `std::sort`, ...
- Features:
 - Number of threads: with `OMP_NUM_THREADS` or `--hpx::threads=N`
 - HW Perf. Counters: PAPI's HL or Likwid's Marker APIs
 - Different (customizable) input sizes and data types
 - Custom NUMA allocator

Variables

- Input sizes: 2^3 to 2^{30} elements \rightarrow 64B to 8GB
- Data type: double and float
- *pSTL-Bench*'s NUMA allocator
- No control of thread and memory placement

Contenders

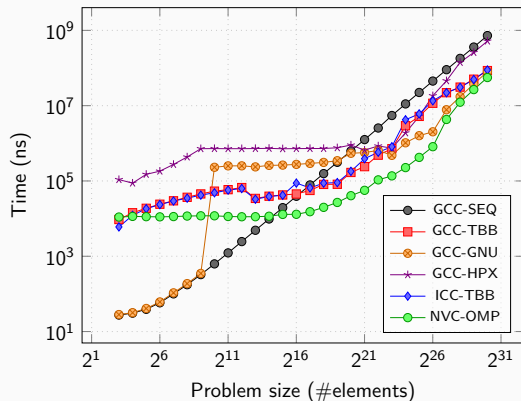
- Compilers: GCC, ICC, NVIDIA HPC SDK
- Backends: GNU (OpenMP), TBB, HPX, Thrust (OMP), CUDA

Experiments

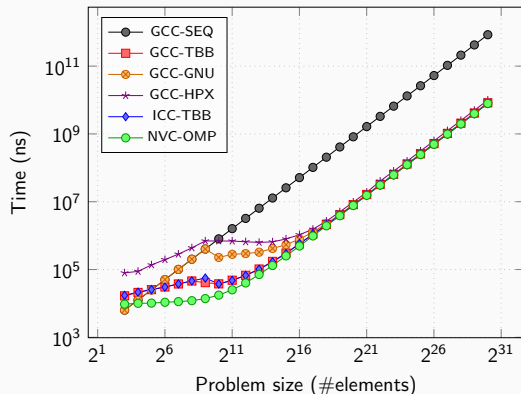
Machine	VSC-5 (Zen 3)	Hydra (Skylake)	Tesla	Ampere
CPU/GPU	AMD EPYC 7713	Intel Xeon 6130F	NVIDIA Tesla T4	NVIDIA Ampere A2
Architecture	Zen 3	Skylake	Turing	Ampere
Sockets NUMA nodes	2 8	2 2	1 1	1 1
Total #cores threads	128 256	32 32	2560 2560	1280 1280
Max. #threads used	128	32	2560	1280
Memory (node / GPU)	512 GiB	48 GiB	16 GiB	8 GiB
Memory (per core)	4 GiB	1.5 GiB	—	—
STREAM BW 1 all core(s) (GB/s)	42.6 249	11.7 135	N/A 264	N/A 172

Results - Execution time scaling

Low computational intensity ($k_{it} = 1$)



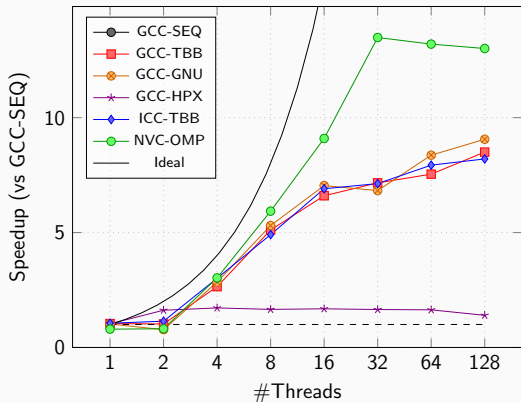
High computational intensity ($k_{it} = 1000$)



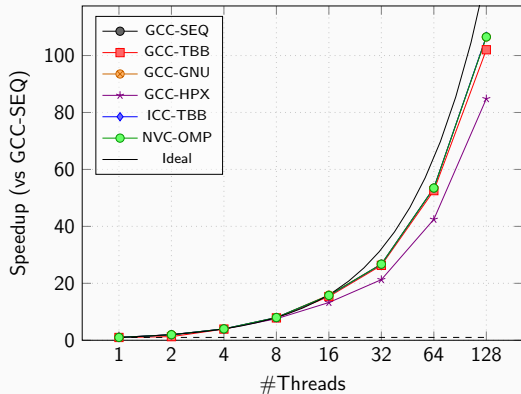
Execution time scaling of `for_each` in **VSC-5 (Zen 3)**. Data type: double. **All** cores are used except for GCC-SEQ. Lower is better.

Results - Speedup

Low computational intensity ($k_{it} = 1$)



High computational intensity ($k_{it} = 1000$)



Strong scaling of `for_each` with 2^{30} doubles in **VSC-5 (Zen 3)**. Higher is better.

Results - HW Counters and Binary Sizes

Executed instructions in 100 calls to `std::for_each` ($k_{it} = 1$) on **Hydra (Skylake)**.

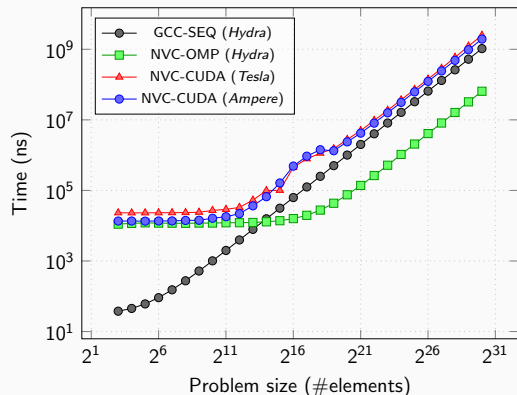
Metric	GCC	GCC	GCC	ICC	NVC
	TBB	GNU	HPX	TBB	OMP
Instructions	1.72T	2.41T	3.83T	1.55T	2.24T
FP scalar	107G	107G	107G	107G	107G
FP 128-bit packed	0	0	0	0	0
FP 256-bit packed	0	0	0	0	0
GFLOP/s	5.41	6.51	4.06	5.02	7.26
Mem. bandwidth (GiB/s)	107.6	116.6	75.6	104.5	119.1
Mem. data volume (GiB)	2128	1925	1850	2151	1762

Binary sizes in **Hydra (Skylake)** and **Tesla**. Lower is better.

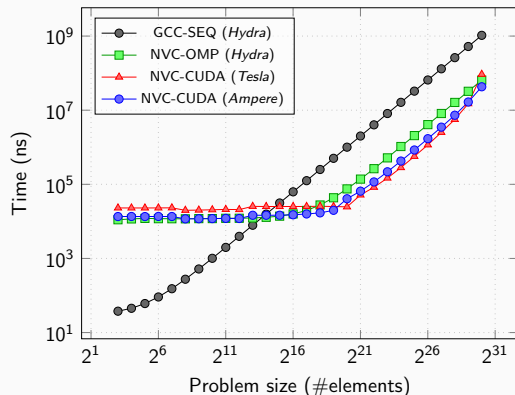
Compiler Backend	Binary size (MiB)
GCC-SEQ	2.5
GCC-TBB	17.2
GCC-GNU	5.3
GCC-HPX	62.0
ICC-TBB	16.6
NVC-OMP	1.8
NVC-CUDA	7.8

Results - GPUs

Continuous H2D and D2H transfers



No H2D and D2H transfers







Execution time scaling of reduce. Data type: float. **All cores** are used except for GCC-SEQ. Lower is better.

Your mileage will vary

- C++ is rapidly moving towards performance portability with execution policies
 - Actual implementations are not so rapid
- *pSTL-Bench* is a tool to evaluate the performance of C++ STL
- Performance is *heavily* dependent on the compiler and backend
- Algorithms in STL are usually memory-bound → Not a huge speedup + careful memory placement
- In GPUs, data transfer is the key → keep data in the device

<https://github.com/parlab-tuwien/pSTL-Bench>

 README  GPL-3.0 license  

pSTL-Bench

pSTL-Bench is a benchmark suite designed to assist developers in evaluating the most suitable parallel STL (Standard Template Library) backend for their needs. This tool allows developers to benchmark a wide variety of parallel primitives and offers the flexibility to choose the desired backend for execution during compile time.

Table of Contents

- [Introduction](#)
- [Features](#)
- [Installation](#)
- [Usage](#)

Exploring Scalability in C++ Parallel STL Implementations

International Conference on Parallel Processing 2024 — ICPP'24

Ruben Laso, Diego Krupitza, and Sascha Hunold

`{laso, krupitza, hunold}@par.tuwien.ac.at`

August 13, 2024

Research Group for Parallel Computing, TU Wien



Informatics

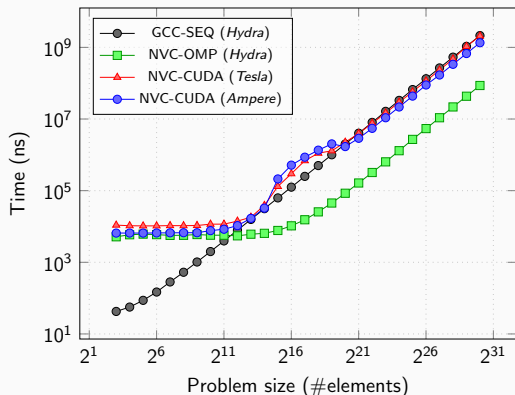
Additional content

Maximum number of threads such that **efficiency is above** 70 % (compared to the seq. execution) for **VSC-5 (Zen 3)**. Problem size is 2^{30} . Higher is better.

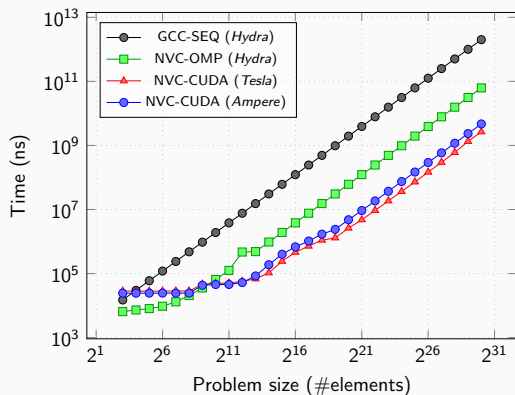
	find	for_each $k_{it} = 1$	for_each $k_{it} = 1000$	inclusive_scan	reduce	sort
GCC-TBB	2	1	128	1	16	8
GCC-GNU	1	1	128	N/A	16	32
GCC-HPX	1	2	16	1	4	4
ICC-TBB	1	4	128	1	1	8
NVC-OMP	4	16	128	1	32	2

Additional content

Low computational intensity ($k_{it} = 1$)



High computational intensity ($k_{it} = 1000$)



Execution time scaling of `for_each`. Data type: `float`. All cores are used except for GCC-SEQ. Lower is better.