Reproducing C++ Multicore and GPU Benchmark Results on Chameleon Cloud

Community Workshop on Practical Reproducibility in HPC

Ruben Laso[1] (ruben.laso.rodriguez@univie.ac.at)
Sascha Hunold[2] (hunold@par.tuwien.ac.at)

Nov 18, 2024

[1]Research Group for Scientific Computing, University of Vienna
[2]Research Group for Parallel Computing, TU Wien

C++ and Performance Portability

C++'s execution policies

- Since C++17, we have *execution policies* to express parallelism
  - `std::execution::<seq,par,par_unseq>`
  - `#include <execution>`
- These policies are applicable to many algorithms in the Standard Library (STL)

```
std::for_each(v.begin(), v.end(), f); // Sequential iteration over elements of v
↪  applying lambda function f
std::for_each(std::execution::seq, v.begin(), v.end(), f); // Sequential
std::for_each(std::execution::par, v.begin(), v.end(), f); // Parallel
std::for_each(std::execution::par_unseq, v.begin(), v.end(), f); // Parallel and
↪  (potentially) vectorized
```

C++ and Performance Portability

C++'s execution policies

- The implementations of the STL might be different:
  - Execution policies are not limited to CPUs!
  - Different backends: OpenMP, TBB, **CUDA**, ...

Research questions

1. **Speedup** and efficiency of parallel STL algorithms?
2. Which is the **best compiler/backend**? GCC vs ICC, TBB vs OpenMP, ...
3. In which cases to use **CPU or GPU backends**?

pSTL-Bench

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

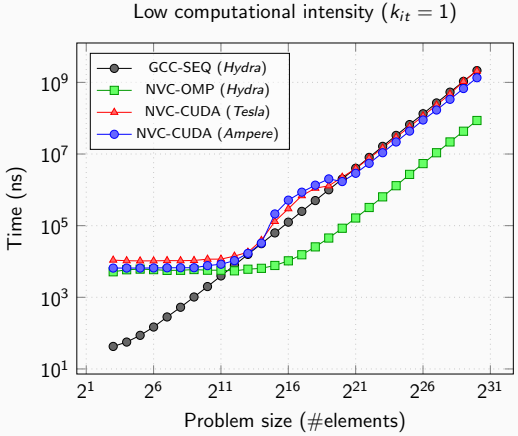
README GPL-3.0 license

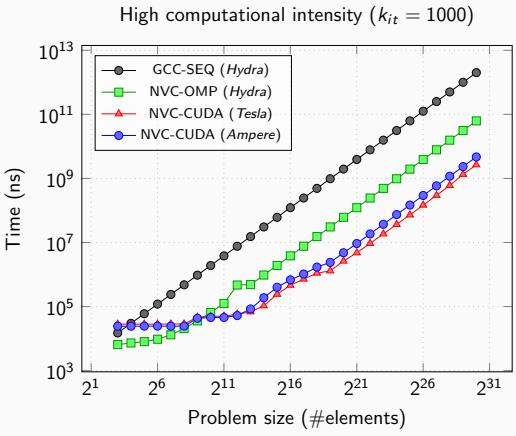
pSTL-Bench

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

pSTL-Bench





Execution time scaling of `std::for_each`. Intel Xeon 6130F vs NVIDIA Tesla T4 vs NVIDIA Ampere A2.

Exploring Scalability in C++ Parallel STL Implementations

Ruben Laso
ruben.laso@tuwien.ac.at
Faculty of Informatics
TU Wien
Vienna, Austria

Diego Krupitza
krupitza@par.tuwien.ac.at
Faculty of Informatics
TU Wien
Vienna, Austria

Sascha Hunold
sascha.hunold@tuwien.ac.at
Faculty of Informatics
TU Wien
Vienna, Austria

Ruben Laso, Diego Krupitza, and Sascha Hunold. "Exploring Scalability in C++ Parallel STL Implementations". In: *Proceedings of the 53rd International Conference on Parallel Processing*. ICPP '24. Gotland, Sweden: Association for Computing Machinery, 2024, pp. 284–293. ISBN: 9798400717932. DOI: 10.1145/3673038.3673065

How to get the three badges?

Aims

- ~~Get the three badges to show off~~
- Be responsible scientists who care about reproducibility
- Have a "one-click" solution to easily reproduce the results
  - Make the life of the reviewers easier
  - https://github.com/parlab-tuwien/pSTL-Bench-ICPP24-ADAE

How to get the three badges?

Aims

- ~~Get the three badges to show off~~
- Be responsible scientists who care about reproducibility
- Have a "one-click" solution to easily reproduce the results
  - Make the life of the reviewers easier
  - https://github.com/parlab-tuwien/pSTL-Bench-ICPP24-ADAE

Three-step process

1. **Software setup**: Chameleon Cloud's Images
2. **Hardware setup**: Chameleon Cloud's Jupyter interface
3. **Experiments execution**: Chameleon Cloud's Jupyter interface

Software setup

ubuntu22-pstl

Image

ID	02e6eff2-e748-4cdc-baa2-0aead7f3ab7b
Name	ubuntu22-pstl
Type	
Status	Active
Size	15.16 GB
Min. Disk	0
Min. RAM	0
Disk Format	QCOW2
Container Format	BARE
Created At	Jun 19, 2024 8:42:14 PM
Updated At	Jun 20, 2024 7:48:03 AM

Chameleon Cloud's Images

1. Start a base image. E.g., Ubuntu 22.04
2. Install the necessary software:
   - Compilers and backends
   - pSTL-Bench
   - Scripts to run the experiments
3. Export the image to be used in the next steps:
   - `sudo cc-snapshot <image_name>`

Hardware setup

Chameleon Clouds's Jupyter interface and CHI

1. Setup variables: project ID, user, image...
2. Create a lease with a `compute_zen3` node
3. Start the `ubuntu22-pstl` image

[8]:
```
# Create a reservation for your nodes
from chi import lease

reservations = []
lease.add_node_reservation(reservations, count=SERVER_COUNT, node_type=NODE_TYPE)
start_date, end_date = lease.lease_duration(days=1)
lease_info = lease.create_lease(f"{USERNAME}-lease", reservations=reservations,
                                start_date="now", end_date=end_date)
lease_id = lease_info.get("id")
my_log('Waiting for the lease to be active...')
active_lease_info = lease.wait_for_active(lease_id)
my_log('Lease is active')
```

Experiments execution

[11]:
```python
# Run all experiments with "00_run_all.sh"
from chi import ssh
from wand.image import Image as WI

figures = [
    'fig2a_for_each_its1.pdf', 'fig2b_for_each_its1000.pdf',
    'fig3a_for_each_its1.pdf', 'fig3b_for_each_its1000.pdf',
    'fig5a_incl_scan.pdf', 'fig5b_incl_scan.pdf',
    'fig7a_sort.pdf', 'fig7b_sort.pdf'
]

with ssh.Remote(computing_fip) as conn:
    conn.run('./00_run_all.sh')
    for fig in figures:
        download_fig(conn, fig)

for fig in figures:
    my_log(fig)
    display(WI(filename=f"pstl-figs/{fig}"))
```

Chameleon Cloud's Jupyter interface

1. Connect with SSH
2. Run the experiments
3. Collect the results
4. **Done!**

Reproduced results

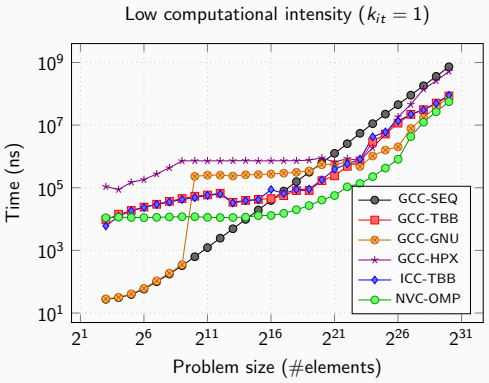
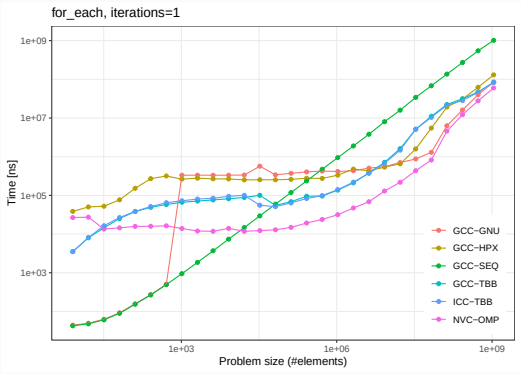


Figure 2a from the paper. AMD EPYC 7713 128-Core (VSC-5)



Reproduced figure in CC. AMD EPYC 7763 64-Core (`compute_zen3`)

Lessons learned using Chameleon Cloud

GPUs

- **Plan ahead if using NVIDIA GPUs**: these nodes are always busy
- **AMD GPUs might be problematic**: no pre-configured images $\rightarrow$ manual setup

Chameleon Cloud's infrastructure

- **Use the Jupyter interface**: best thing ever
  - Programmatic hardware setup, execution of experiments and results collection
  - *Easy to reproduce!*

Lessons learned using Chameleon Cloud

Chameleon Cloud's infrastructure (II)

- **Custom images are a win-win**: save time and effort for everyone
  - For authors: reuse the same image while preparing artifacts, new experiments, ...
  - For reviewers: easy to reproduce the results
  - For the community: easy to use the same setup for new experiments
  - *Creating the snapshot can take a while!*

General advise

- **Automate the reproduction of the experiments**: have a "one-click" solution
  - Enumerate scripts: `01_compile.sh`, `02_fig1.sh`, ...
  - One script to rule them all: `00_run_all.sh`

Conclusions

How to get the three badges?

- Make it easy for the reviewers to reproduce the results:
  - Make use of **Chameleon Cloud's infrastructure** to provide an easy access to the software and hardware
  - One script to rule them all!

Lessons learned using Chameleon Cloud

- Issues with GPUs: availability (NVIDIA) or setup (AMD)
- Automation provided by Chameleon Cloud is key for reproducibility
  - Make use of Jupyter interface and custom images

Reproducing C++ Multicore and GPU Benchmark Results on Chameleon Cloud

Community Workshop on Practical Reproducibility in HPC

Ruben Laso[1] (ruben.laso.rodriguez@univie.ac.at)
Sascha Hunold[2] (hunold@par.tuwien.ac.at)

Nov 18, 2024

[1]Research Group for Scientific Computing, University of Vienna
[2]Research Group for Parallel Computing, TU Wien