GNU Parallel for Optimal Utilization of HPC Resources on Frontier Supercomputer

Ketan Maheshwari, NCCS, Oak Ridge National Laboratory

with: Bill Arndt (NERSC), Ahmad Maroof Karimi, Junqi Yin, Fred Suter, Seth Johnson, Rafael Ferriera da Silva

OLCF User Meeting'24

Sep 11, 2024

Acknowledgements

This work is funded by the U.S. DOE Office of Science. This research used resources of the OLCF at ORNL, which is supported by the Office of Science of the U.S. DOE under Contract No. DE-AC05-00OR22725.

Overview

- Introduction
- Capabilities: Scaling and stress tests on Frontier and Perlmutter
- GNU Parallel use cases
  - Science Applications: Data Processing, ML, GPU and HMMSearch
  - Operational: Massive Data Transfer, Systems' Performance Analysis
- Summary / Conclusions

Talk is motivated by three aspects of large-scale computing:
- Utilization of resources (CPU, GPU, NVMe)
- Opportunistic parallelism, which is found ~everywhere
- Application to real applications

What is GNU Parallel?

- Linux terminal tool to **run processes in parallel**
- A low-friction, featureful parallelizing tool
- Available via the GNU project: gnu.org/software/parallel
- Simple to use, easy to learn
- Available as a software module on Frontier

```
module load parallel
```

Usage: In the small

```
parallel -X rm {} ::: *.tar.xz

find . -iname '*.docx’ | parallel -X pandoc {} -o {.}.pdf

cat tasklist.txt | parallel

find . -name ‘*.pdb' | parallel gzip --best

find ./gitrepos -maxdepth 1 | parallel git -C {} pull

parallel -N0 -j32 "yes>/dev/null" ::: {0..32}
```

github.com/ketancmaheshwari/pearc24tut

Usage: In the large

- On Frontier
  - Scaling across the compute nodes
  - Practical application runs and resource usage
- On Perlmutter
  - Stress test: processes creation rate per second
  - How about containers?

Scaling Benchmark on Frontier: Outline

- Slurm parameters
- srun call as **one task per node**
- Xfer results to lustre from NVMe

- Invokes Payload script via GNU Parallel
- Invokes with inputs as command line parameter
- **Evenly distributes inputs to nodes**

- Application invocation
- Receives inputs from Driver
- May have other inputs optionally

Job Definition

```
#!/bin/bash

#SBATCH -J 9000nodes_128j

#SBATCH -o %x-%j.out

#SBATCH -e %x-%j.err

#SBATCH -t 2:00:00

#SBATCH -N 9000

#SBATCH -C nvme

srun --no-kill --ntasks-per-node=1 --wait=0 ./driver.sh ./input.txt > /mnt/bb/ketan2/out.txt

cat /mnt/bb/ketan2/out.txt # cat to stdout on lustre
```

Driver

Invoked as ./driver.sh input.txt
input.txt has **(nodes X 128)** lines

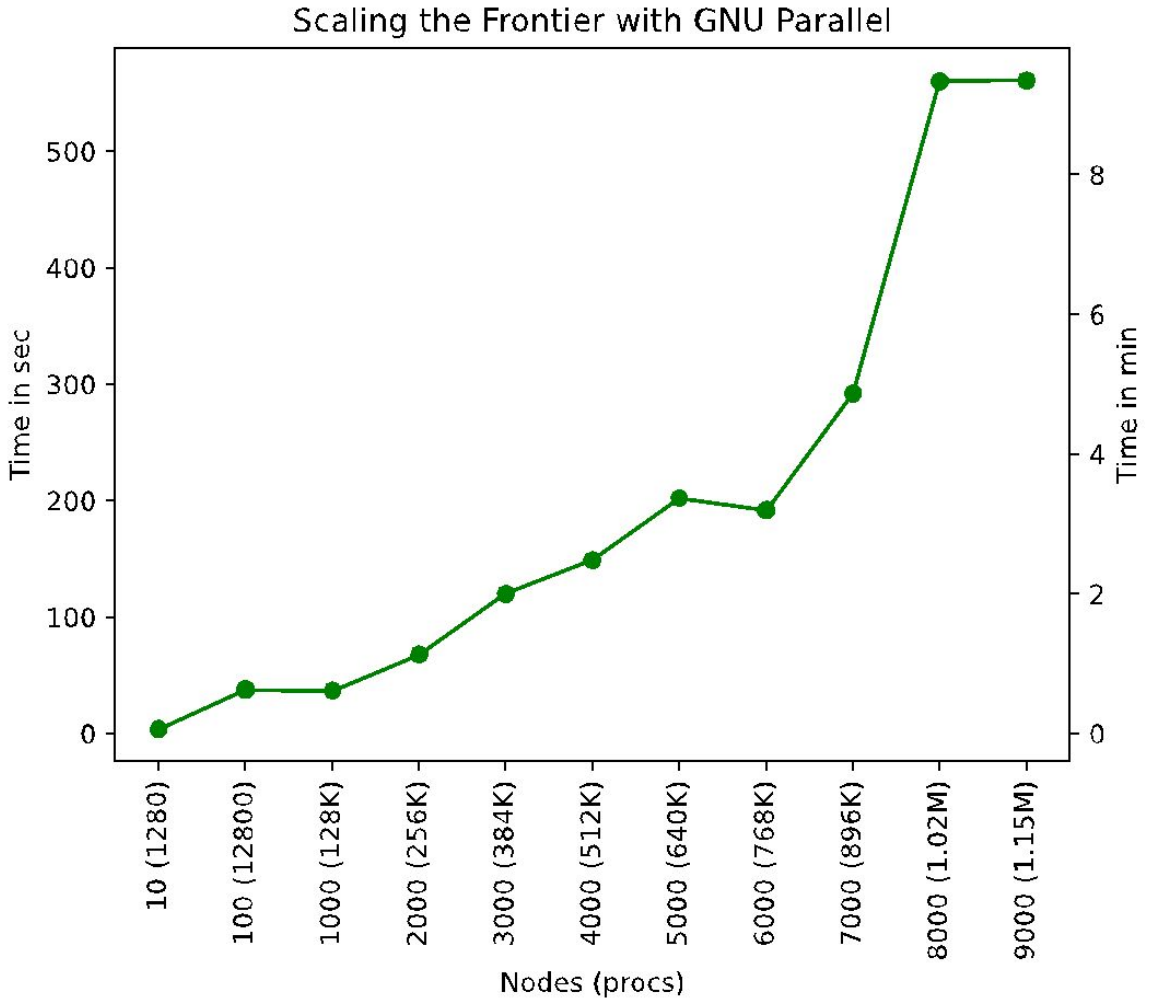
```
cat $1 | \                              #1 pipe the inputs to awk
awk -v NNODE="$SLURM_NNODES" \          #2 env var to awk var
    -v NODEID="$SLURM_NODEID" \         #3 env var to awk var
    'NR % NNODE == NODEID' | \          #4 cur line pass if true
parallel -j128 ./payload.sh {}          #5 128 payload.sh in ||
```

Payload

```
#!/bin/bash

H="$(hostname)"
echo "Arg is $1, Ran on node $H at $(date +%s)"
```

Performance / overhead

- Nodes X 128 Tasks
- Single phase parallel execution of tasks
- Lustre I/O excluded from measurements
- Time is the difference between smallest and largest value of $(date +%s) in the output

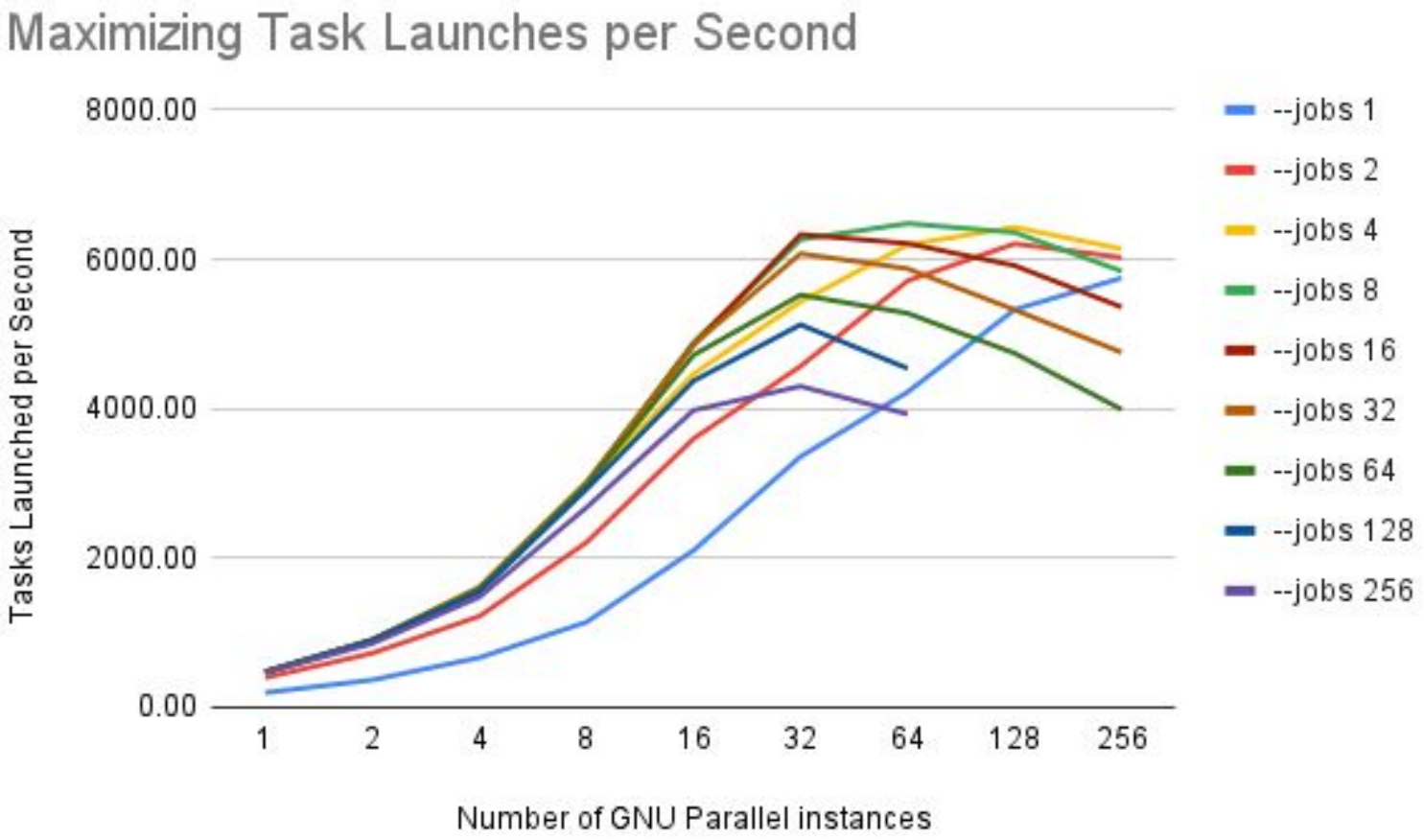


Frontier Scaling Analysis

Half of the processes completed in less than a minute, and 75\% completed in less than two minutes with 8,000 nodes.

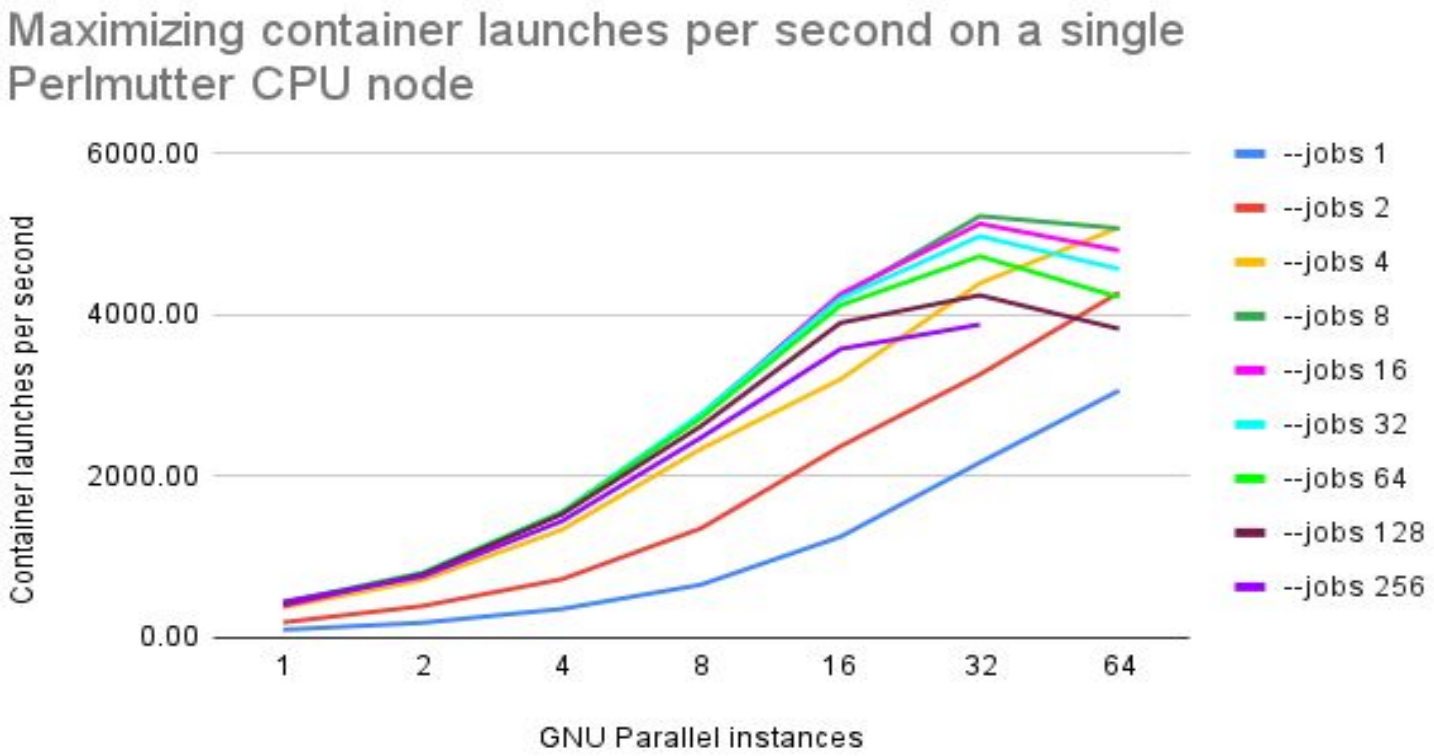
Greater variance was observed in 9,000-node run

Process creation rates for multiple GNU parallel instances



- Process launch rate upper bound ~6400 processes per second
  - Process duration could be as low as ~40 milliseconds and still maintain node utilization

Container launch rates on Perlmutter



- The measured container launch rate **upper bound is ~5200 processes per second**
  - Shifter container startup overhead relative to "bare metal" is only **~19%**

Diverse Use Cases: Overview

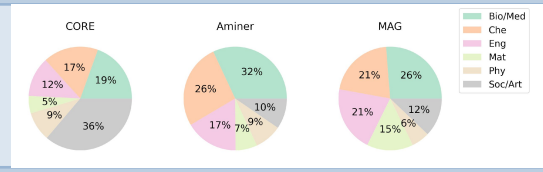
- Applications
  1. Forge: Large Scale Pre-Training Models
  2. Darshan Log Processing and Archival / Publication
  3. Celeritas GPU based Monte Carlo Simulation in HEP
- Operational
  1. PoliMOR filesystem management performance analysis
  2. Massive data migration with rsync

FORGE: Pre-Training Open Foundation Models for Science

POC: Junqi Yin and Feiyi Wang

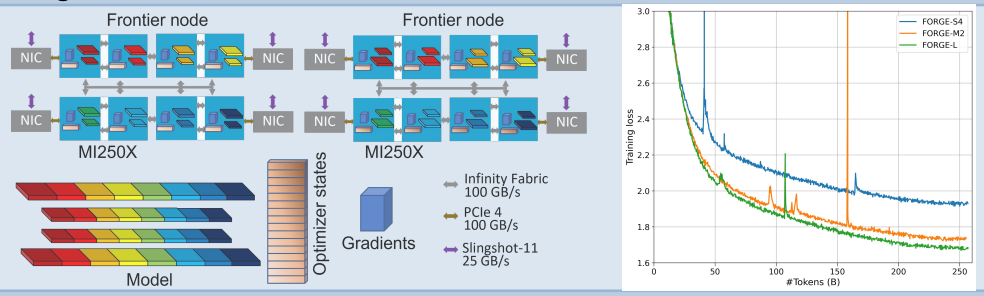
Research Objective	• The project aims for an end-to-end examination of effectiveness of large language models (LLMs) in scientific research, including their scaling behavior and computational requirements on Frontier at ORNL.
Scientific Achievement	• The team has developed for release to the scientific community a suite of open foundation models called FORGE with about 22B parameters using 257B tokens from over 200M scientific articles, with performance either on par or superior to other state-of-the-art comparable models. • The team has demonstrated the use and effectiveness of FORGE on scientific downstream tasks, including using model embeddings as a knowledge representation (e.g., domain and phase classification, and energy regression).
Significance and Impact	• This study provides practical solutions for building and using LLM-based foundation models targeting scientific research and use cases. • The team's approach and findings have significant implications for developing Language Models that can better understand and interpret scientific language and thus support scientific research and discovery.

Scientific Texts

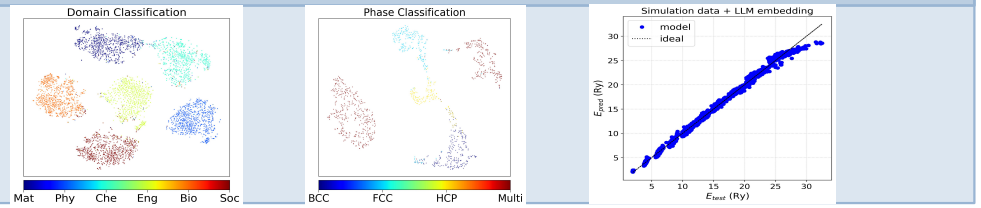


Source	#Docs (M)	# Tokens (B)	Size (GB)
CORE	52.3	225	764
Aminer	47.1	10	55
MAG	103.8	20	108
SCOPUS	6.3	1.5	6.5
ArXiv	2.2	0.8	3.4
Total	212	257	937

Training



Scientific Downstream Tasks



Publication(s) for this work:

1. J. Yin, S. Dash, J. Gounley, F. Wang, G. Tourassi. Evaluation of pre-training large language models on leadership-class supercomputers. *Journal of Supercomputing*, 2023.
2. J. Yin, S. Dash, F. Wang, M. Shankar. FORGE: Pre-Training Open Foundation Models for Science, *SC'23*, 2023.

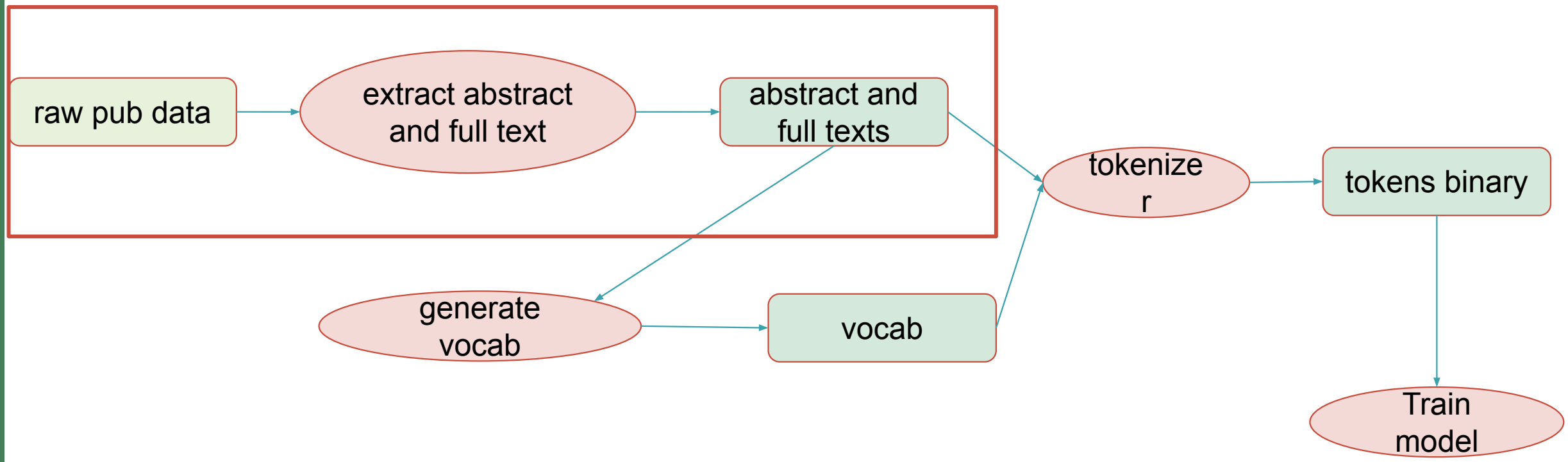
Slide Courtesy: Junqi Yin (ORNL)

About the Data and Process

- Publications Data from Aminer, CORE, Scopus, Arxiv, MAG
- A collection of ~200M json files totaling 10T+

- Extract Full text + abstracts
- Filter out non-English language content
- Filter out non-English characters

Application Workflow: Forge ML



An example Publications' Record


```
{"doi": null, "coreId": "147282919", "oai": "oai:NSYSU:etd-0709101-171951",
"magId": null, "identifiers": [], "title": "Exploring eCommerce Strategies and
Business Models for Gold & Jewelry Industry in Taiwan", "authors": ["Chin-Chang,
Lin"], "enrichments": {"references": [], "citationCount": null, "documentType":
{"type": null, "confidence": null}}, "contributors": ["Jeng Bingchiang", "Wang
Sha-Yain", "Tu Yi-Min"], "datePublished": "2001-07-09", "abstract": null,
"downloadUrl": null, "fullTextIdentifier": null, "pdfHashValue": null,
"publisher": "NSYSU", "journals": [], "language": null, "relations": [], "year":
2001, "topics": ["eMarketplace", "Gold & J", "Electronic commerce", "B2B"],
"subjects": ["text"], "fullText": null, "urls":
["http://etd.lib.nsysu.edu.tw/ETD-db/ETD-search/view_etd?URN=etd-0709101-171951",
"http://etd.lib.nsysu.edu.tw/ETD-db/ETD-search/view_etd?URN=etd-0709101-171951"],
"issn": null}
```

An example abstract

....
"abstract": " Abstract This work was developed in order to validate a simplified computer simulation method for application in the drawing processes. The results of tensile tests on AISI 1004, AISI 1020 steel and copper were compared to those of computer simulations performed using the Deform-3D TM software. Each specimen was assumed to be an elasto-plastic material and was meshed with 50,000 finite elements. Young's modulus and Poisson's ratio were the only material properties considered in the elastic region; the parameters of Hollomon's equation were used in the plastic region up to constriction. A strain rate of 1x10-3 s-1 was applied during the simulations. In the plastic region, up to the point of constriction, the curves obtained from the simulations showed reasonable correspondence to those determined from physical testing. However, the former diverged from the latter in the elastic and elastic-plastic transition regions. This divergence indicates that microstructural factors may have a greater influence in the transition region than in the plastic region. Moreover, the correlation obtained in the plastic region indicates that the proposed method has the potential to be applied in drawing processes. </div",...

But Sometimes we have

```
Das Copyright bleibt bei\nden Herausgebern oder sonstigen
Rechteinhabern. Als Nutzer sind Sie sind nicht dazu berechtigt, eine
Lizenz zu \u00fcbertragen, zu\ntransferieren oder an Dritte weiter
zu geben.\nDie Nutzung stellt keine \u00dcbertragung des
Eigentumsrechts an diesem Dokument dar und gilt vorbehaltlich der
folgenden\nEinschr\u00e4nkungen:\nSie m\u00fcssen auf
s\u00e4mtlichen
```

OR

пиченням у місцевих бюджетах менш \nкількості податків, сплачених саме комерційними банками, чер \nвіддаленість від столичного регіону [97, с.79].\nНаявні диспропорції потребують розробки адекватно \nінструментарію вимірювання, оцінки і обґрунтування напрямів \nусунення. Однією з основних причин низьк \nконкурентоспроможності вітчизняної банківської системи є \nнедостатня капіталізація, оскільки ефективне використані \nфінансових ресурсів значною мірою залежить саме від сукупної \nкапіталу комерційних банків, який визначає потенціал банківськ \nсистеми.\nВідтак вітчизняна банківська система не спроможні \nзабезпечувати потреби зростаючої економіки при існуючому ріні

Job Definition

```
#!/bin/bash
#SBATCH -J extractfromjson
#SBATCH -o results.txt
#SBATCH -e %x-%j.err
#SBATCH -t 8:00:00
#SBATCH -p batch
#SBATCH -N 64
#SBATCH --ntasks-per-node 1

srun --no-kill --ntasks-per-node=1 --wait=0 ./driver.sh
”/lustre/orion/proj-shared/stf008/junqi-work/data”
```

Driver

```
#!/bin/bash

module load parallel
if [[ -z "${SLURM_NODEID}" ]]; then
echo "need \$SLURM_NODEID set”
exit
fi

if [[ -z "${SLURM_NNODES}" ]]; then
echo "need \$SLURM_NNODES set”
exit
fi

find $1 -type f -iname ‘*.json’ |

awk -v NNODE="$SLURM_NNODES" -v NODEID="$SLURM_NODEID" 'NR % NNODE == NODEID' |

parallel -j128 ./payload.sh {}
```

Payload

```
jq -r 'select(.abstract != null).abstract,
select(.fullText != null).fullText' $1 |

./langdetect.pl   |

perl -CAS -pe 's/\P{^Script: Han}//g;' -pe
's/\P{^Script: Hangul}//g;' -pe 's/\P{^Script: Cyrillic}//g;'
-pe 's/\P{^Script: Hiragana}//g;' -pe 's/\P{^Script:
Arabic}//g;' -pe 's/\P{^Script: Thai}//g;’ |

tr -s `,;:".’
```

Use Case: Darshan Log Processing and Archival

- Parse Darshan logs from Summit and process:
  - Identify the application, Anonymize
  - Compress and Publish in parquet format
  - Summit data available for **2019-2023**

- Use case demonstrates:
  - Conciseness of GNU Parallel
  - Data prefetching to NVMe
  - Workflow Construction
  - Compute node co-scheduling

Before GNU Parallel

```
#!/bin/bash

#SBATCH --mem=0
#SBATCH -t 20:00:00
#SBATCH -N 1
#SBATCH -J archive-darshan
#SBATCH -o o-archive.MACHINE
#SBATCH -e e-archive.MACHINE

module load cray-python


IFS=,
months='1,2,3,4,5,6,7,8,9,10,11,12'
apps_lst='3'

months=($months)
apps_lst=($apps_lst)
counter=0

for month in ${months[@]}; do
  apps=${apps_lst[counter]}
  app=0
  while [[ $app -lt ${apps} ]] ; do
    echo "Month: "${month} " App: " ${app}
    srun -N1 -n1 -c1 --exclusive python3 darshan_arch_storage.py ${month} ${app} &
    sleep 0.2
    ((app++))
  done;
done;

wait
```

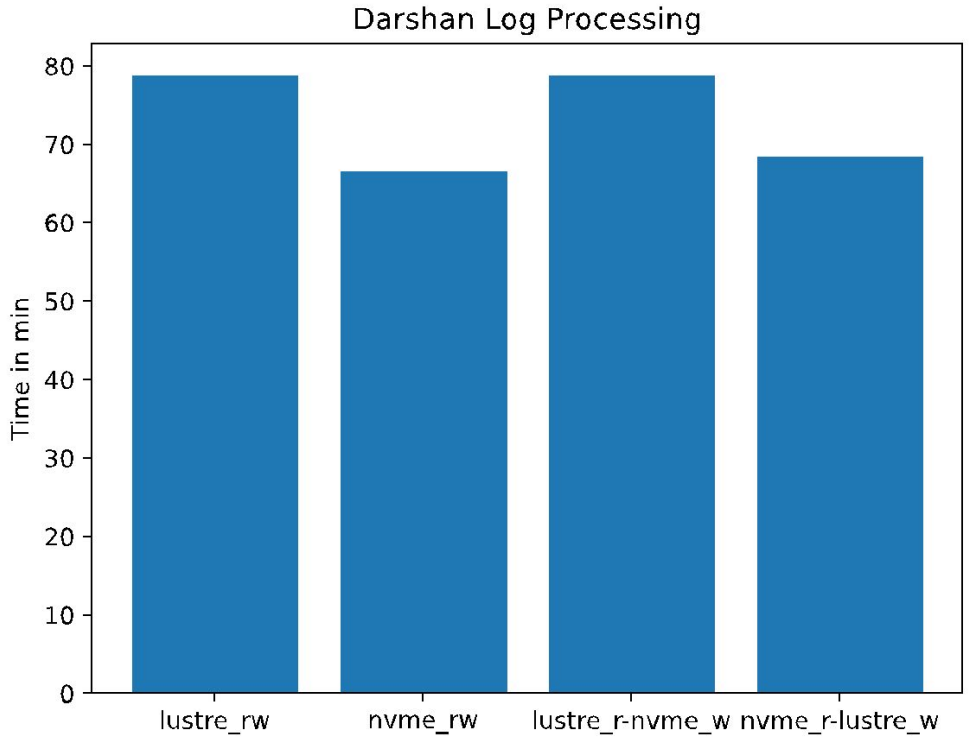
After GNU Parallel

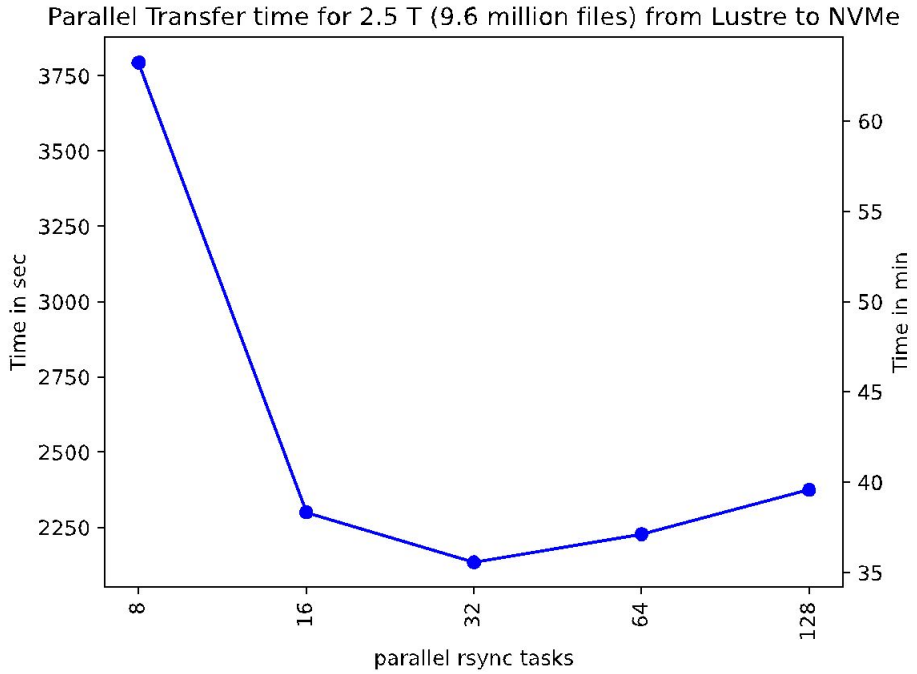
```
#!/bin/bash

#SBATCH --mem=0
#SBATCH -t 2:00:00
#SBATCH -N 1
#SBATCH -J archive-darshan
#SBATCH -o o-archive.MACHINE
#SBATCH -e e-archive.MACHINE

module load cray-python/3.11.5
module load parallel

parallel -j36 python3 ./darshan_arch_storage.py ::: {1..12} ::: {0..2}
```

Opportunities : Using NVMe and Lustre for I/O

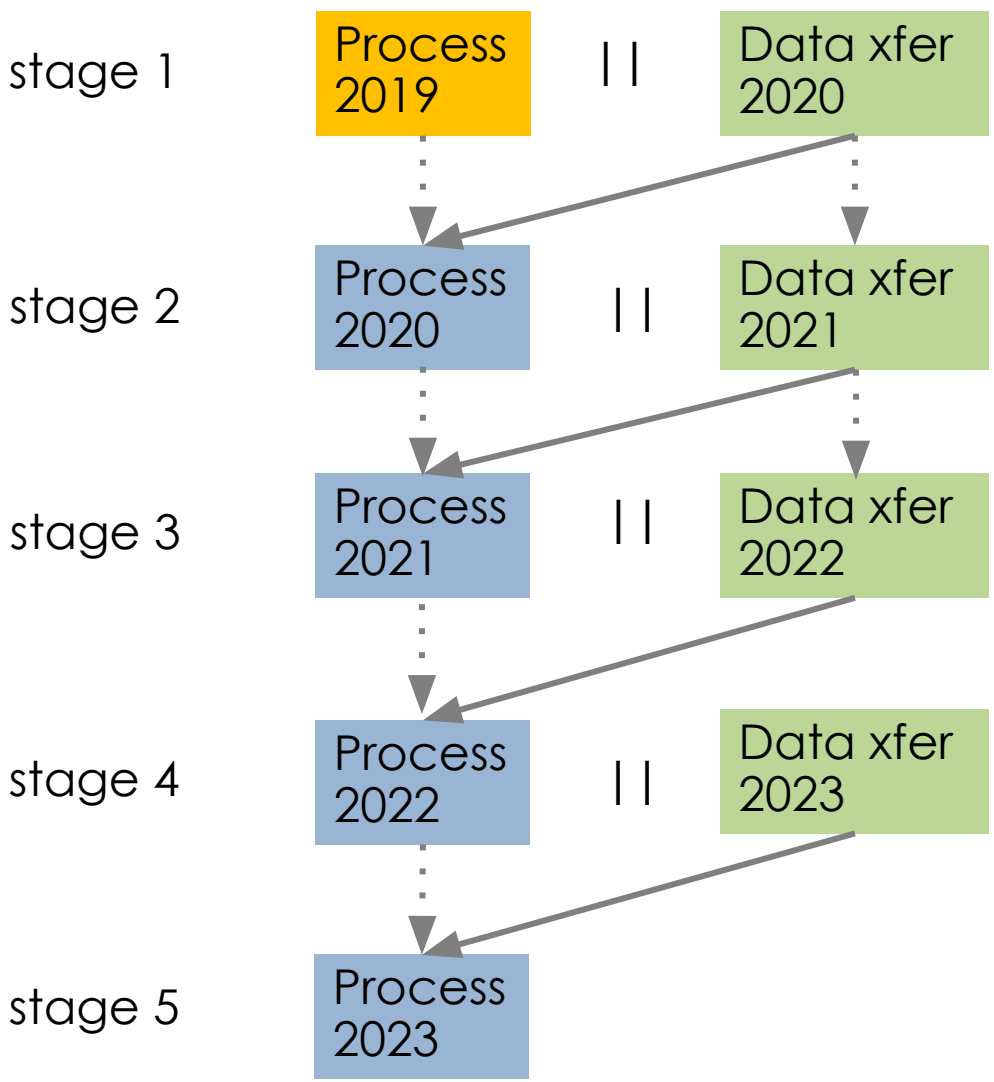
- Four possible I/O configurations
  - Lustre Read / Write
  - NVME Read / Lustre Write (most useful)
  - NVME Read / Write (best performance)
  - Lustre Read / NVME Write

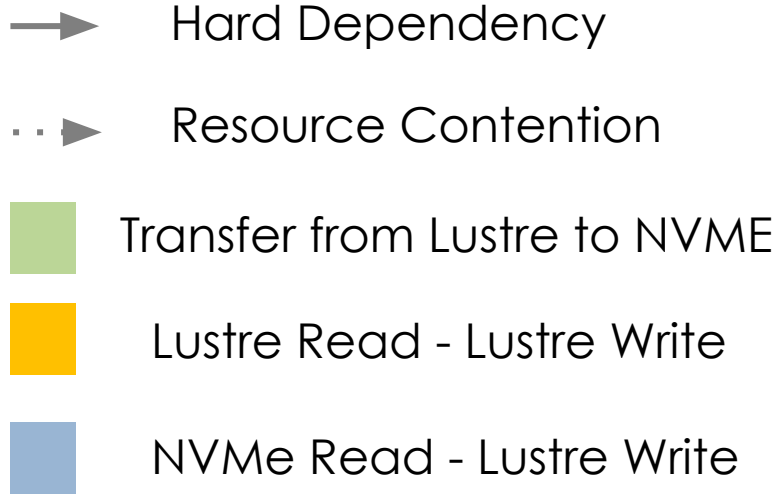
- Cost: Data needs to be copied to NVMe
  - We find that the data movement performance is best when using 32 parallel rsync tasks





Workflow to process five years' data





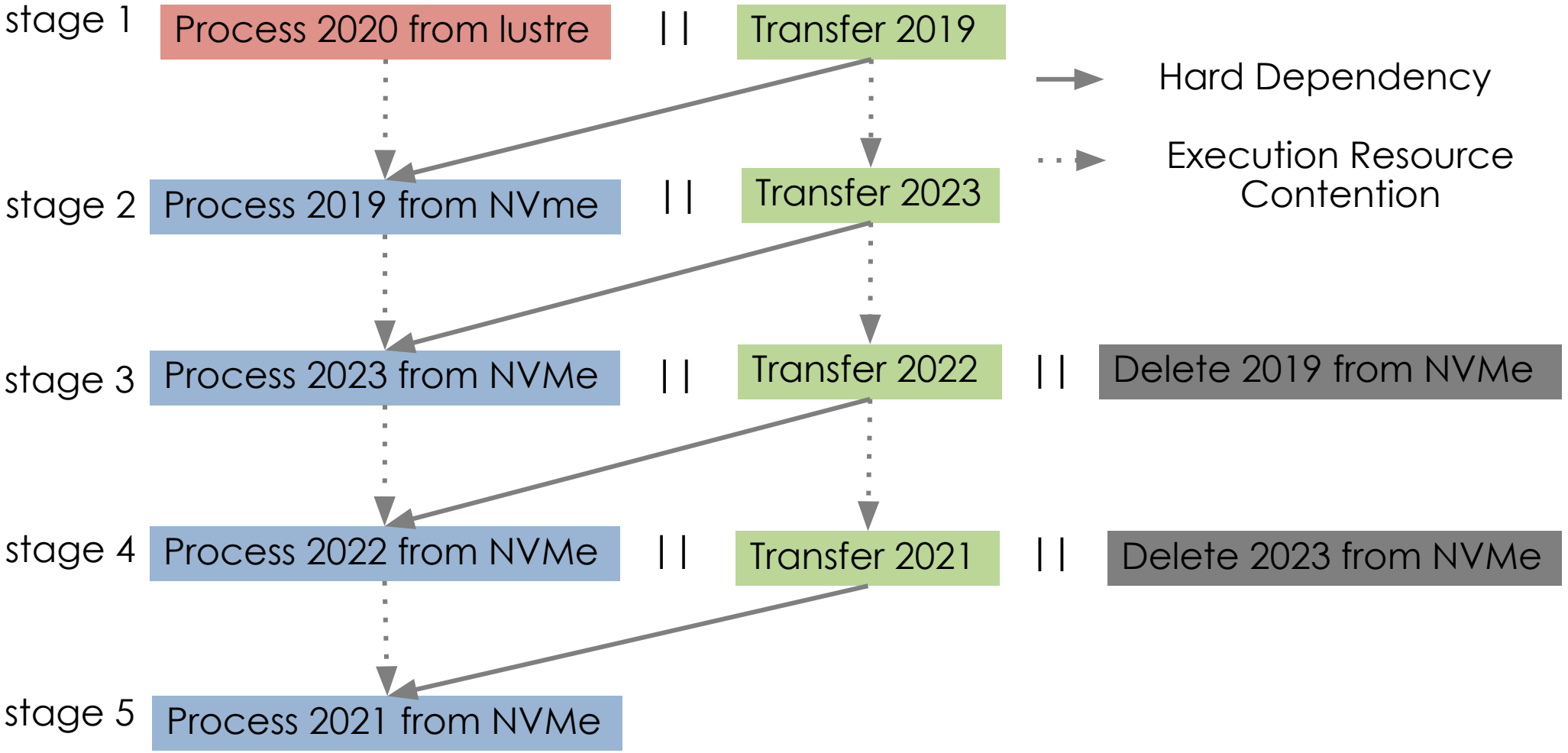
However, there is a problem!

- The data sizes of the yearly datasets need to be considered:
  2019 - 2.4 T
  2020 - 3.3 T
  2021 - 2.5 T
  2022 - 1.3 T
  2023 - 0.7 T
- The total space available on NVMe is 1.9X2 = 3.8 T

So, some adjustments are needed

- Based on the Data sizes, only a few pairs of datasets may live together on the NVMe:
  2019 + 2022 = 2.4 + 1.3 = 3.7 T
  2019 + 2023 = 2.4 + 0.7 = 3.1 T
  2021 + 2022 = 2.5 + 1.3 = 3.8 T [Close, so a bit risky]
  2021 + 2023 = 2.5 + 0.7 = 3.2 T
  2022 + 2023 = 1.3 + 0.7 = 2.0 T
- Additionally, need to ensure the processed data is removed to accommodate new incoming data

So, our Workflow now is ...



Implementation -- overall workflow and stage3.in

```
#!/bin/bash

module load cray-python/3.11.5
module load parallel

source venv/bin/activate



cat stage1.in | parallel

cat stage2.in | parallel

cat stage3.in | parallel

cat stage4.in | parallel

cat stage5.in | parallel
```

```
parallel -j36 python3
darshan_archival_storage_app_nvmer_lustrew.py
::: {1..12} ::: {0..2} ::: 2023


find
/lustre/orion/proj-shared/stf218/data/lake/summ
it_darshan/2022  -type f | parallel -j32 -X
rsync -R -Ha {} /mnt/bb/ketan2/


find
/mnt/bb/ketan2/lustre/orion/proj-shared/stf218/
data/lake/summit_darshan/2019 -type f |
parallel -j32 rm -f {}
```

Application Use Case: Celeritas

- GPU Monte Carlo HEP particle detector simulation code
  - Started in 2020
- Multi-institution effort
  - ORNL, Fermilab, ANL, LBNL, U. of Warwick, and U. of Virginia
- Mostly funded through **DOE ASCR & HEP**

- github.com/celeritas-project

Celeritas R&D Report: Accelerating Geant4. https://doi.org/10.2172/2281972

Courtesy: Stefano Tognini, Seth Johnson, Fred Suter

Multiple nodes, 8 gpus per node

job def

```
#!/bin/bash

#SBATCH -J multinodes
#SBATCH -o %x-%j.out
#SBATCH -e %x-%j.err
#SBATCH -t 1:00:00
#SBATCH -p batch
#SBATCH -N 4
#SBATCH -C nvme

srun --no-kill --ntasks-per-node=1
--wait=0 ./driver.sh
```

driver.sh

```
#!/bin/bash

module load parallel
module load amd/5.2.0
export CELER_LOG=warning


ls *.inp.json | \

awk -v NNODE="$SLURM_NNODES" -v NODEID="$SLURM_NODEID" \
'NR % NNODE == NODEID' | \
parallel -j8 HIP_VISIBLE_DEVICES='$(({%} - 1))'
../celeritas/build-ndebug/bin/celer-sim {} ">"
/mnt/bb/ketan2/{}.out


mv /mnt/bb/ketan2/*.out ./celeritas-run/celer-regression/
```

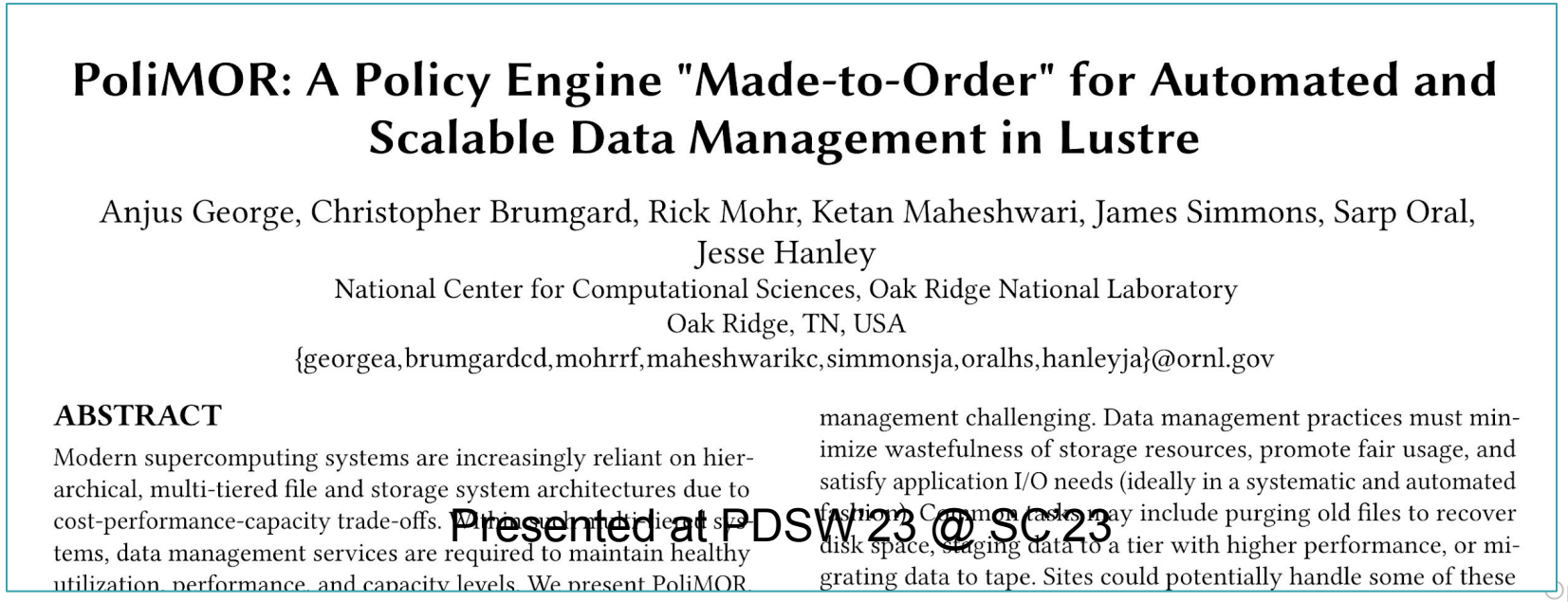
Celeritas Performance

Weak scaling on Frontier GPU nodes with Celeritas demonstrates linear performance and efficient GPU utilization.

The variance in execution time was less than 9 seconds across runs on 10 to 100 nodes, each running 8 GPU processes per node.

Use Case: PoliMOR

- PoliMOR is an in-house software for Lustre Filesystem Management at OLCF
- A distributed, agent-based system that performs file purging and migration based on the center policies



PoliMOR: A Policy Engine "Made-to-Order" for Automated and Scalable Data Management in Lustre

Anjus George, Christopher Brumgard, Rick Mohr, Ketan Maheshwari, James Simmons, Sarp Oral, Jesse Hanley
National Center for Computational Sciences, Oak Ridge National Laboratory
Oak Ridge, TN, USA
{georgea,brumgardcd,mohrrf,maheshwarikc,simmonsja,oralhs,hanleyja}@ornl.gov

ABSTRACT

Modern supercomputing systems are increasingly reliant on hierarchical, multi-tiered file and storage system architectures due to cost-performance-capacity trade-offs. Without automated storage systems, data management services are required to maintain healthy utilization, performance, and capacity levels. We present PoliMOR

management challenging. Data management practices must minimize wastefulness of storage resources, promote fair usage, and satisfy application I/O needs (ideally in a systematic and automated fashion). Common tasks may include purging old files to recover disk space, staging data to a tier with higher performance, or migrating data to tape. Sites could potentially handle some of these

Presented at PDSW'23 @ SC23

PoliMOR Design

- Multiple instances of four agents run in parallel, communicating via a message queuing system
- Policy Agent: Describes the purge and migration policies
- Scan Agent: Scans the file system to collect file metadata
- Purge Agent: Deletes files on policy match
- Migration Agent: Migrates data on policy match

Objective: Measure Performance of PoliMOR

- Generate files with arbitrary attributes (timestamps, size, etc.)
- Run the agents concurrently, incrementally across multiple independent nodes
- Measure process timings over a testbed cluster of 12 nodes running a lustre filesystem

GNU Parallel Vignettes for PoliMOR Performance Testing

Change file's attributes (eg. timestamp) en-masse:

```
parallel -S rage4 -j32 "touch -d '-1 week'\
./{1}/file.{2}" ::: {0..63} ::: {1..3000}
```

8 instances of scan agent across 8 nodes:

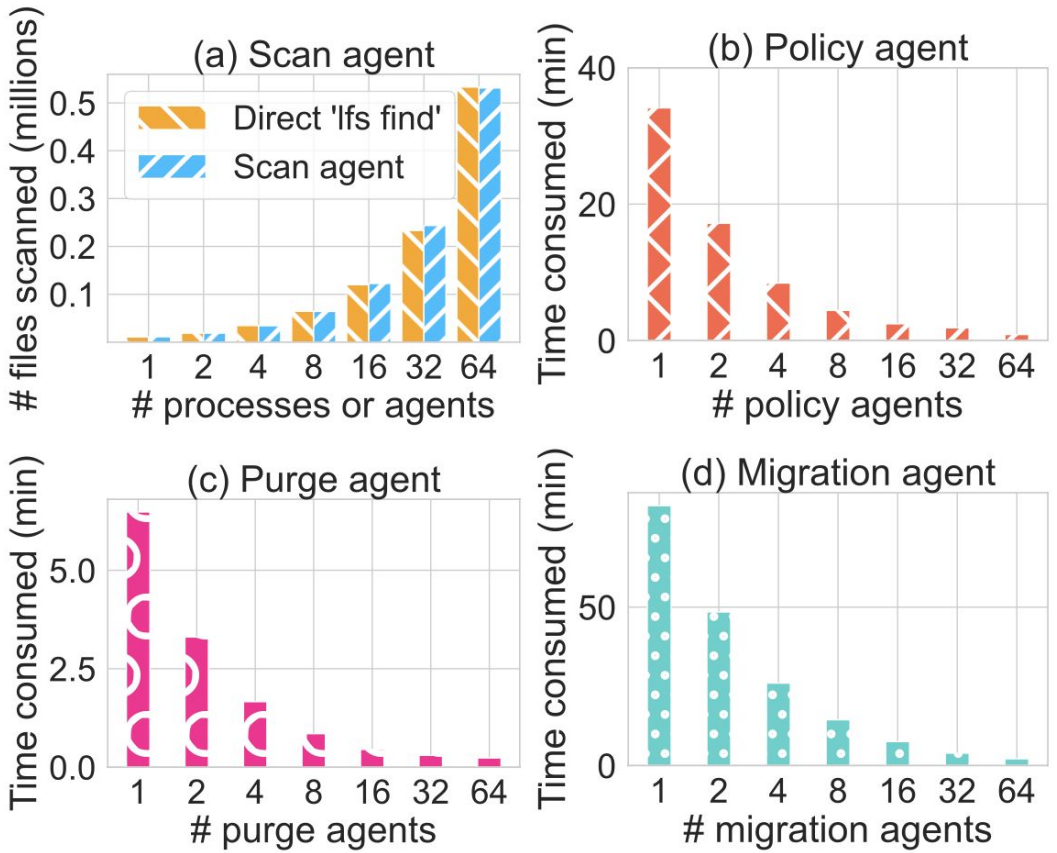
```
parallel -S rage{5..12} ${scan_cmd} :::
${scan_path}/{44..51} >> scanperf.8proc.8node
```

Post migration requests to nats message queue:

```
parallel -S rage4 -j30 'nats -s rage2:4222 \
pub migration.files.request --count 1 \
"{\"path\":\"/lustre/crius/migagent/{1}/file.{2}\"}"
'\ ::: {0..63} ::: {1..3000}
```

PoliMOR Results

- PoliMOR performance results for agents
- Increasing number of agents run in parallel over nodes with a target number of lustre files
- Their performance measured in elapsed time processing the files
- Scan agent compared with baseline lfs find command



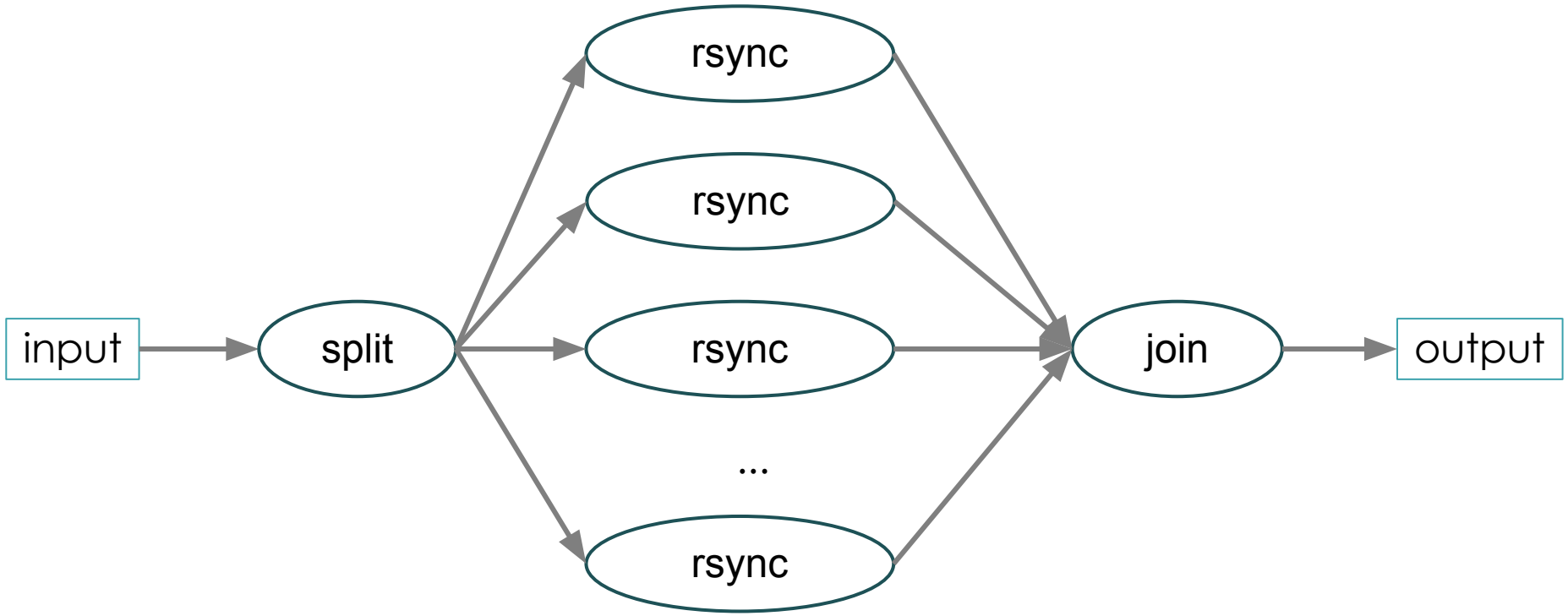
Use Case: Massive Data Migration with rsync and GNU Parallel

- Migrate ~1 PB of project data from Alpine into Lustre
- Hard deadline due to Alpine decommissioning
- Globus resources too busy
- Via an 8-node (non-Globus) DTN cluster

GNU Parallel rsync driver

```
#!/bin/bash

cd /gpfs/alpine/proj-shared/projname/

find . -type f |                                          \

awk -v NNODE="$SLURM_NNODES" -v NODEID="$SLURM_NODEID" \

'NR % NNODE == NODEID' |                                  \

parallel --tmpdir /tmp --compress -j16 -X rsync -R -Ha {}
/lustre/orion/proj-shared/projname/
```

For the case where file is too large



split - process - join

split - parallel rsync - join

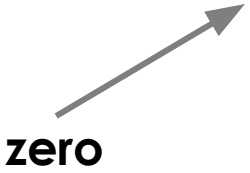
```
split -n32 largefile

parallel -j32 -X rsync --remove-source-files -Ha
./{} /dest/ ::: x*

cat /dest/x* > /dest/largefile && rm -f /dest/x*
```

Summary

- GNU parallel as an effective tool for efficient utilization of diverse HPC resources
- Not just an *ad hoc* utility but a mainstream instrument for large scale task parallel computing on scalable systems
- It is an enabler, try it! ☺

Thank you for your time!
Questions?

km0@ornl.gov



Task distribution to nodes, an example

Let $1 be `input.txt` with 5 lines and job size be 4 nodes

=> SLURM_NNODES = NNODE = 4 ..... (line: `awk -v NNODE="$SLURM_NNODES"`)

=> SLURM_NODEID = NODEID = 0, 1, 2, 3 ..... (line: `-v NODEID="$SLURM_NODEID"`)

Awk's **`NR`** variable: 1, 2, ..., 5
True / False ..... (line: `'NR % NNODE == NODEID'`)

1 % 4 = 1 => true for NODEID 1 => line 1 will be on Node1
2 % 4 = 2 => true for NODEID 2 => line 2 will be on Node2
3 % 4 = 3 => true for NODEID 3 => line 3 will be on Node3
4 % 4 = 0 => true for NODEID 0 => line 4 will be on Node0
5 % 4 = 1 => true for NODEID 1 => line 5 will be on Node1