Performance-portability and task parallelism for Lattice Field Theory

Bartosz Kostrzewa*, Stefan Krieg†, Estela Suarez‡

*High Performance Computing & Analytics Lab
†Helmholtz Zentrum für Strahlen und Kernphysik
‡Institut für Informatik
Rheinische Friedrich-Wilhelms-Universität Bonn
†‡Juelich Supercomputing Center, Forschungszentrum Juelich

Colour meets Flavour – Innovative Algorithms Lab Kick-off meeting
2024.03.20

Technical Challenges in Lattice Field Theory

⟨NuMeriQS⟩

- Assume that sampling problems in lattice field theory can be tempered or overcome:
  - critical slowing down
  - ergodicity issues in the presence of multiple phases
  - volume scaling of generative flow models
  - multi-level sampling

Technical Challenges in Lattice Field Theory

⟨NuMeriQS⟩

- Assume that sampling problems in lattice field theory can be tempered or overcome:
  - critical slowing down
  - ergodicity issues in the presence of multiple phases
  - volume scaling of generative flow models
  - multi-level sampling
- What are the major technical challenges facing moonshot-type calculations?

Technical Challenges in Lattice Field Theory

⟨NuMeriQS⟩

- Assume that sampling problems in lattice field theory can be tempered or overcome:
  - critical slowing down
  - ergodicity issues in the presence of multiple phases
  - volume scaling of generative flow models
  - multi-level sampling
- What are the major technical challenges facing moonshot-type calculations?
  - **HPC heterogeneity**: need write-once run-anywhere (as far as possible).

Technical Challenges in Lattice Field Theory

⟨NuMeriQS⟩

- Assume that sampling problems in lattice field theory can be tempered or overcome:
  - critical slowing down
  - ergodicity issues in the presence of multiple phases
  - volume scaling of generative flow models
  - multi-level sampling
- What are the major technical challenges facing moonshot-type calculations?
  - **HPC heterogeneity**: need write-once run-anywhere (as far as possible).
  - Strong scalability: most efficient algorithms don't scale well (maybe they don't need to).

Technical Challenges in Lattice Field Theory

⟨NuMeriQS⟩

- Assume that sampling problems in lattice field theory can be tempered or overcome:
  - critical slowing down
  - ergodicity issues in the presence of multiple phases
  - volume scaling of generative flow models
  - multi-level sampling
- What are the major technical challenges facing moonshot-type calculations?
  - **HPC heterogeneity**: need write-once run-anywhere (as far as possible).
  - Strong scalability: most efficient algorithms don't scale well (maybe they don't need to).
  - **Complex observables** lead to hard-to-debug nested loops with horrible inter-dependencies and low re-use potential.

Technical Challenges in Lattice Field Theory

⟨NuMeriQS⟩

- Assume that sampling problems in lattice field theory can be tempered or overcome:
  - critical slowing down
  - ergodicity issues in the presence of multiple phases
  - volume scaling of generative flow models
  - multi-level sampling
- What are the major technical challenges facing moonshot-type calculations?
  - **HPC heterogeneity**: need write-once run-anywhere (as far as possible).
  - Strong scalability: most efficient algorithms don't scale well (maybe they don't need to).
  - **Complex observables** lead to hard-to-debug nested loops with horrible inter-dependencies and low re-use potential.
  - **Lack of task parallelism**: lots of untapped resources which are inaccessible with current LFT frameworks.

Technical Challenges in Lattice Field Theory

⟨NuMeriQS⟩

- Assume that sampling problems in lattice field theory can be tempered or overcome:
  - critical slowing down
  - ergodicity issues in the presence of multiple phases
  - volume scaling of generative flow models
  - multi-level sampling

- What are the major technical challenges facing moonshot-type calculations?
  - **HPC heterogeneity**: need write-once run-anywhere (as far as possible).
  - Strong scalability: most efficient algorithms don't scale well (maybe they don't need to).
  - **Complex observables** lead to hard-to-debug nested loops with horrible inter-dependencies and low re-use potential.
  - **Lack of task parallelism**: lots of untapped resources which are inaccessible with current LFT frameworks.
  - **Storage**: we cannot store millions of gauge configurations → might need on-the-fly calculations of **everything**.

Technical Challenges in Lattice Field Theory

⟨NuMeriQS⟩

- Assume that sampling problems in lattice field theory can be tempered or overcome:
  - critical slowing down
  - ergodicity issues in the presence of multiple phases
  - volume scaling of generative flow models
  - multi-level sampling

- What are the major technical challenges facing moonshot-type calculations?
  - **HPC heterogeneity**: need write-once run-anywhere (as far as possible).
  - Strong scalability: most efficient algorithms don't scale well (maybe they don't need to).
  - **Complex observables** lead to hard-to-debug nested loops with horrible inter-dependencies and low re-use potential.
  - **Lack of task parallelism**: lots of untapped resources which are inaccessible with current LFT frameworks.
  - **Storage**: we cannot store millions of gauge configurations → might need on-the-fly calculations of **everything**.
  - Programmer productivity: need high-performance backends with easy to use frontends for students (and for ourselves).

Technical Challenges in Lattice Field Theory

⟨NuMeriQS⟩

- Assume that sampling problems in lattice field theory can be tempered or overcome:
  - critical slowing down
  - ergodicity issues in the presence of multiple phases
  - volume scaling of generative flow models
  - multi-level sampling

- What are the major technical challenges facing moonshot-type calculations?
  - **HPC heterogeneity**: need write-once run-anywhere (as far as possible).
  - Strong scalability: most efficient algorithms don't scale well (maybe they don't need to).
  - **Complex observables** lead to hard-to-debug nested loops with horrible inter-dependencies and low re-use potential.
  - **Lack of task parallelism**: lots of untapped resources which are inaccessible with current LFT frameworks.
  - **Storage**: we cannot store millions of gauge configurations → might need on-the-fly calculations of **everything**.
  - Programmer productivity: need high-performance backends with easy to use frontends for students (and for ourselves).
  - Physics students: not enough training in software engineering / HPC / modern C++ (for this kind of work)

HPC heterogeneity

⟨NuMeriQS⟩

Need to target different architectures

- Situation bound to get worse with future specialized hardware & Modular Supercomputing Architecture
  - multi-purpose chiplet designs (EPI)
  - in-memory computing
  - neuromorphic devices
  - tensor & stencil accelerators, large FPGAs

HPC heterogeneity

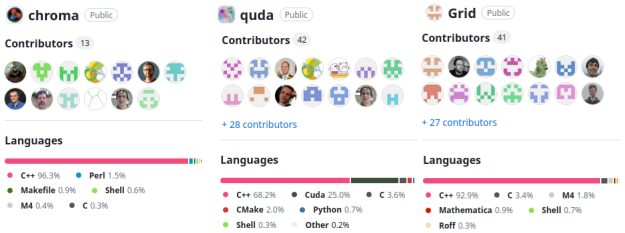
⟨NUMERIQS⟩

Need to target different architectures

intel AMD NVIDIA RISC-V epi European Processor Initiative arm FUJITSU A64FX™

- Situation bound to get worse with future specialized hardware & Modular Supercomputing Architecture
  - multi-purpose chiplet designs (EPI)
  - in-memory computing
  - neuromorphic devices
  - tensor & stencil accelerators, large FPGAs

Existing libraries with some performance-portability



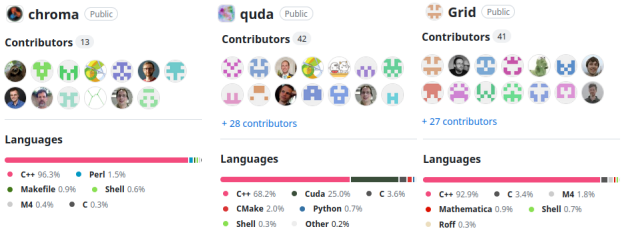
HPC heterogeneity

⟨NuMeriQS⟩

Need to target different architectures

- Situation bound to get worse with future specialized hardware & Modular Supercomputing Architecture
  - multi-purpose chiplet designs (EPI)
  - in-memory computing
  - neuromorphic devices
  - tensor & stencil accelerators, large FPGAs

Existing libraries with some performance-portability



Future directions explored in NuMeriQS B02

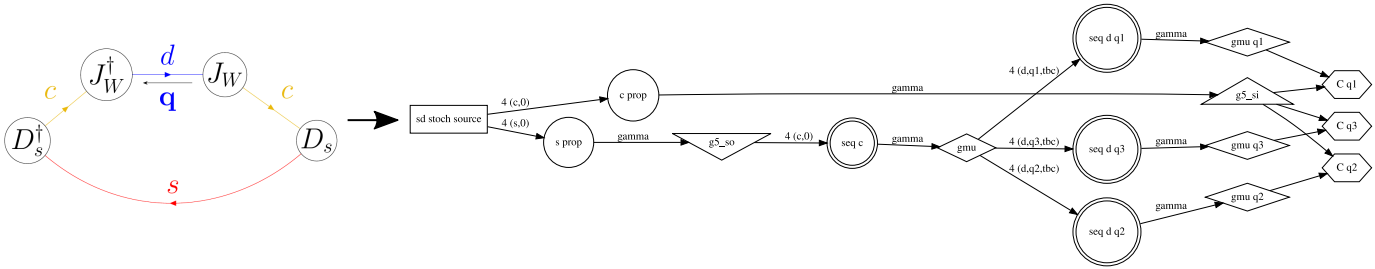
LQFT base layer: abstractions for memory space, memory access patterns and execution space, parallel patterns, asynchronicity via futures, etc.

- **std::execution**
- **std::mdspan**
- **std::async**, **std::future**
- kokkos

Complex observables and nested loops

⟨NuMeriQS⟩

- Simple observable: two current insertions with momentum transfer between meson initial and final states
  - lots of unused parallelism at the inversion / contraction stage (different flavours, different momenta, different Dirac structures)

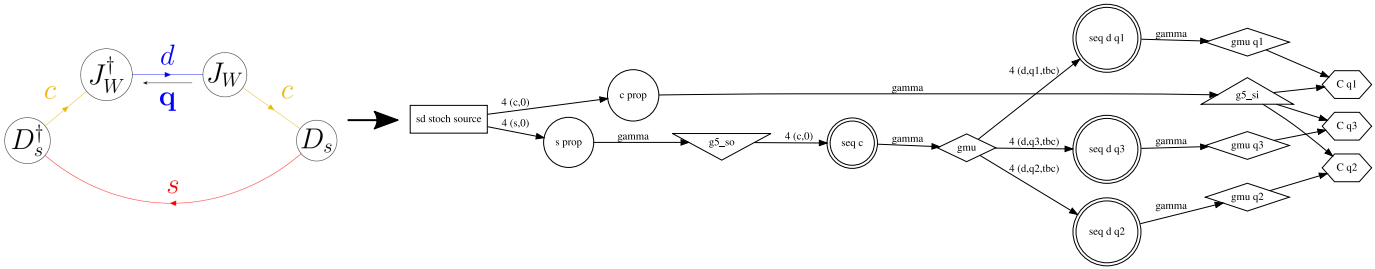


- How to exploit this without writing really complicated nested loops with tons of conditionals?

Complex observables and nested loops

⟨NuMeriQS⟩

- Simple observable: two current insertions with momentum transfer between meson initial and final states
  - lots of unused parallelism at the inversion / contraction stage (different flavours, different momenta, different Dirac structures)

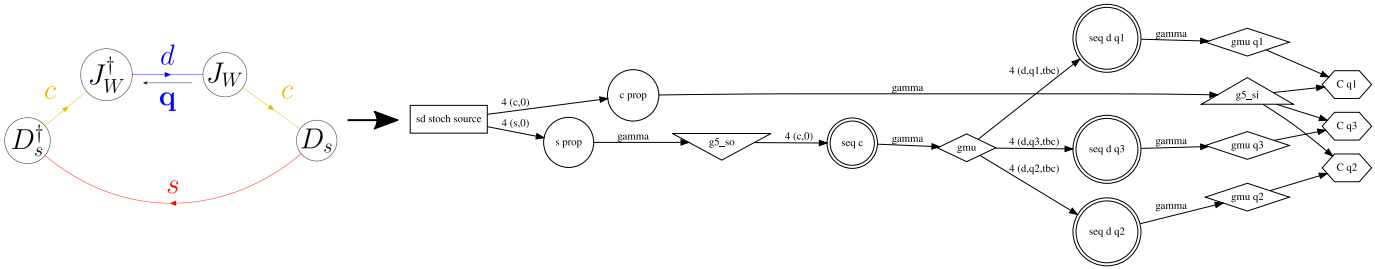


- How to exploit this without writing really complicated nested loops with tons of conditionals?
  - **Collections of observables as forests of directed acyclic graphs**

Complex observables and nested loops

⟨NuMeriQS⟩

- Simple observable: two current insertions with momentum transfer between meson initial and final states
  - lots of unused parallelism at the inversion / contraction stage (different flavours, different momenta, different Dirac structures)

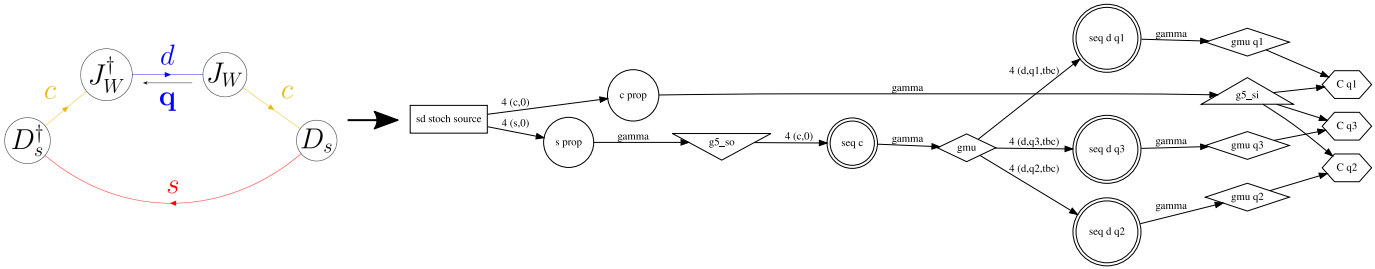


- How to exploit this without writing really complicated nested loops with tons of conditionals?
  - **Collections of observables as forests of directed acyclic graphs**
  - Edges carry computational cost

Complex observables and nested loops

⟨NuMeriQS⟩

- Simple observable: two current insertions with momentum transfer between meson initial and final states
  - lots of unused parallelism at the inversion / contraction stage (different flavours, different momenta, different Dirac structures)

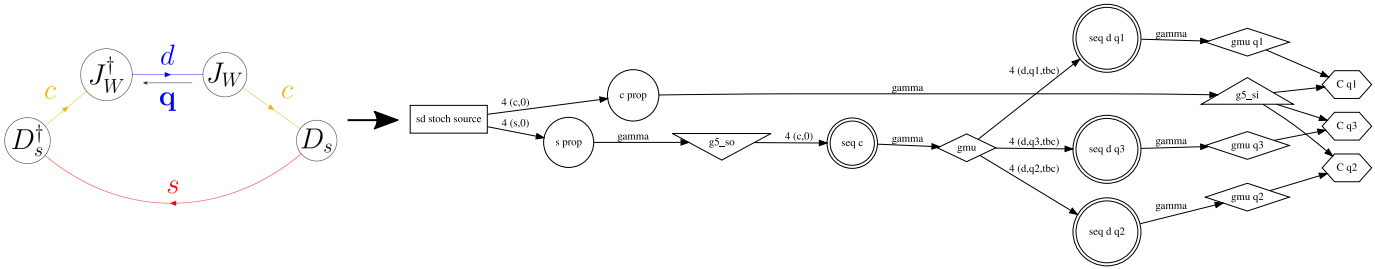


- How to exploit this without writing really complicated nested loops with tons of conditionals?
  - **Collections of observables as forests of directed acyclic graphs**
  - Edges carry computational cost
  - Vertices (may) carry computational and memory cost

Complex observables and nested loops

⟨NuMeriQS⟩

- Simple observable: two current insertions with momentum transfer between meson initial and final states
  - lots of unused parallelism at the inversion / contraction stage (different flavours, different momenta, different Dirac structures)



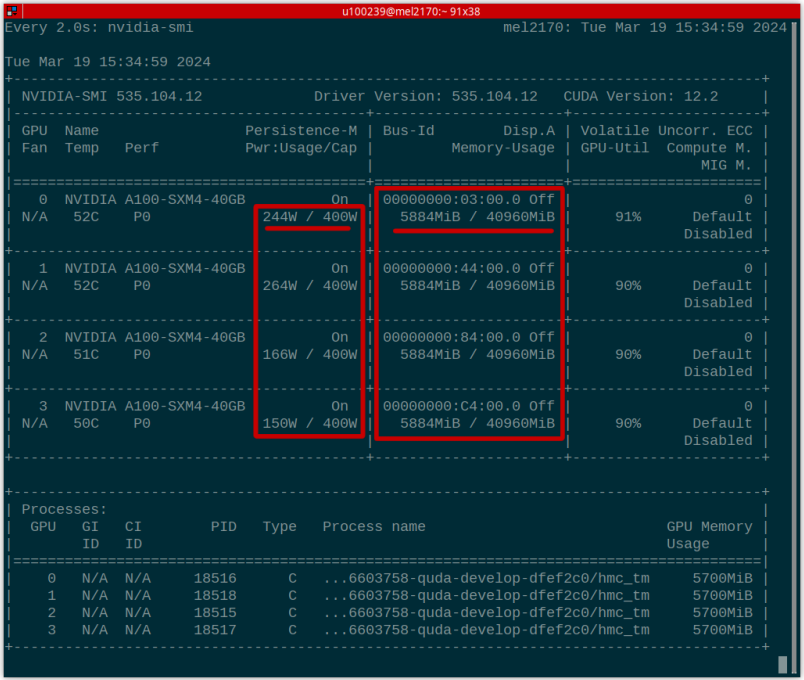
- How to exploit this without writing really complicated nested loops with tons of conditionals?
  - **Collections of observables as forests of directed acyclic graphs**
  - Edges carry computational cost
  - Vertices (may) carry computational and memory cost

⇒ Optimise calculation under machine constraints

  - all dependencies taken care of automatically, no more nested loops, just descend down the graph hierarchy
  - can trade memory or storage for computation or vice-versa

Task parallelism

⟨NuMeriQS⟩

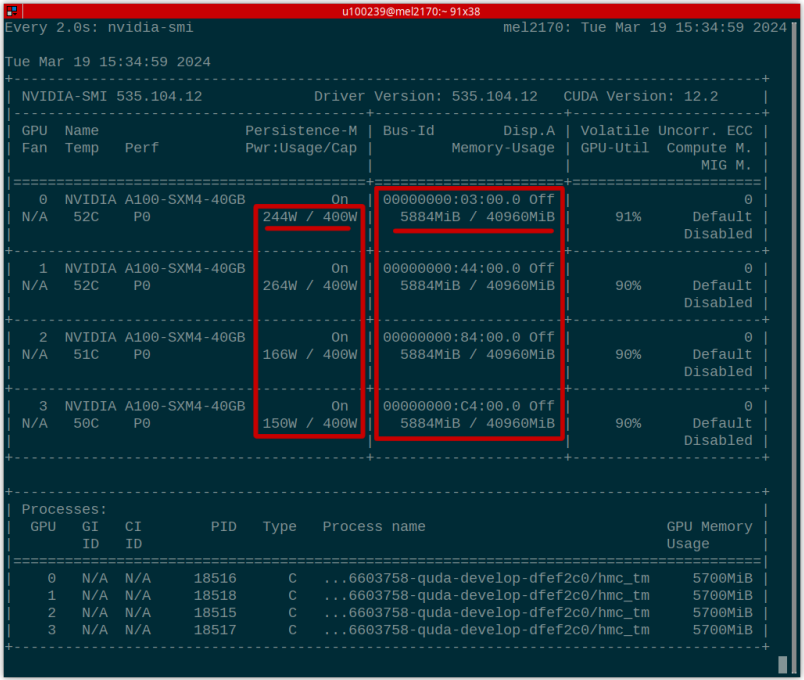


HMC run on 4×A100 nodes at strong scaling limit

- Lots of free resources when running at strong-scaling limit.

Task parallelism

⟨NuMeriQS⟩

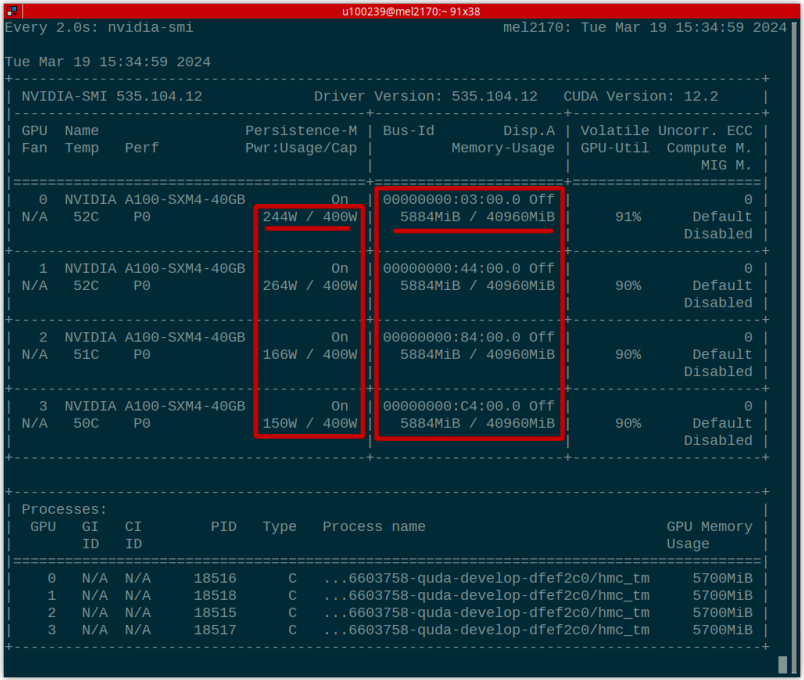


HMC run on 4×A100 nodes at strong scaling limit

- Lots of free resources when running at strong-scaling limit.
- GPU and CPU idle at different times.
  - Wouldn't it be neat to **fully** use the CPU on a Grace-Hopper node?

Task parallelism

⟨NuMeriQS⟩

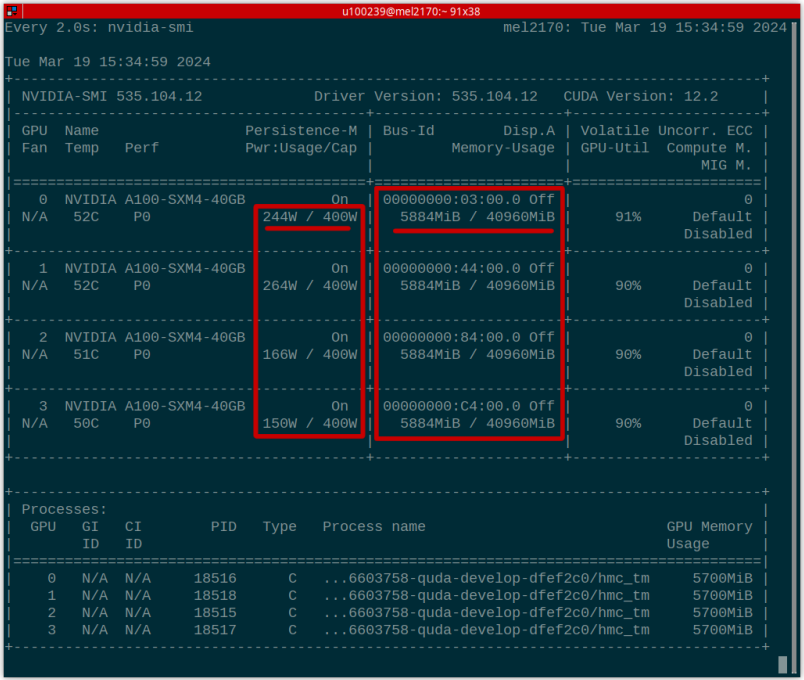


HMC run on 4×A100 nodes at strong scaling limit

- Lots of free resources when running at strong-scaling limit.
- GPU and CPU idle at different times.
  - Wouldn't it be neat to **fully** use the CPU on a Grace-Hopper node?
- Would like to exploit this to run calculations in idle gaps, additional GPU streams.

Task parallelism

⟨NuMeriQS⟩



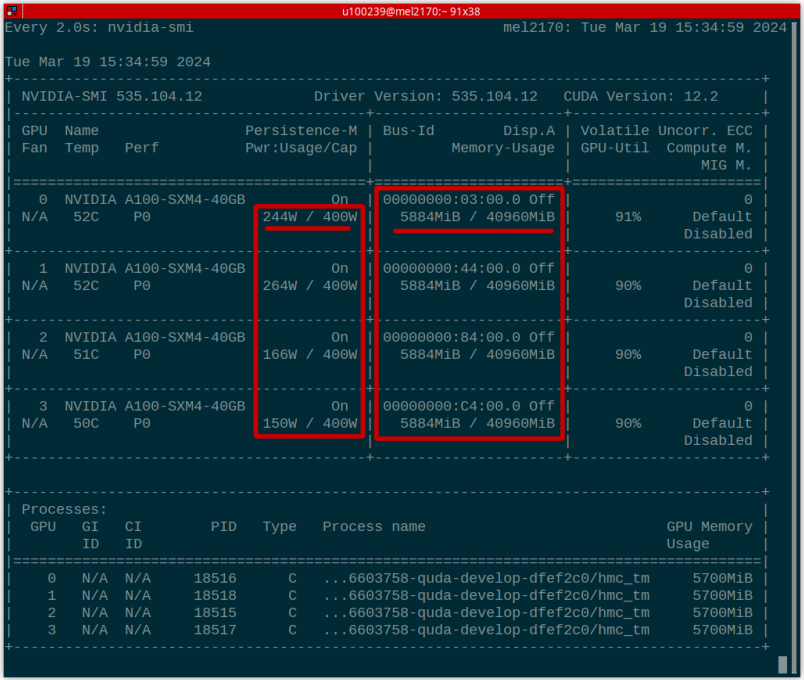
HMC run on 4×A100 nodes at strong scaling limit

- Lots of free resources when running at strong-scaling limit.
- GPU and CPU idle at different times.
  - Wouldn't it be neat to **fully** use the CPU on a Grace-Hopper node?
- Would like to exploit this to run calculations in idle gaps, additional GPU streams.

⇒ Need suitable programming model to express task parallelism, work stealing, etc.

Task parallelism

⟨NuMeriQS⟩



HMC run on 4×A100 nodes at strong scaling limit

- Lots of free resources when running at strong-scaling limit.
- GPU and CPU idle at different times.
  - Wouldn't it be neat to **fully** use the CPU on a Grace-Hopper node?
- Would like to exploit this to run calculations in idle gaps, additional GPU streams.

⇒ Need suitable programming model to express task parallelism, work stealing, etc.

! Must be rather coarse-grained, existing asynchronous many task systems (likely) not suitable.

Summary

⟨NuMeriQS⟩

- Assumption: sampling problems in lattice QFT will be tempered sufficiently
  - produce millions of large volume gauge configurations
  - impossible to store, need to calculate (almost) **everything** on the fly

⇒ Need: task parallelism, DAG representation of observables and dependencies, performance-portable kernels

Issue	Performance-portability	Task parallelism	loops → DAGs	Productivity layer
HPC heterogeneity	✔	–	–	–
Strong scaling	–	✔	✔	–
Wasted resources	✔	✔	✔	–
Complex observables	–	✔	✔	maybe
Storage	–	✔	✔	–
use ML frameworks	–	–	–	✔
Overwhelmed students	✖	✖	✖	maybe