



Scientific Computing in Rust

High-Performance Computing for Quantum Field Theory: A Case for Rust in Monte Carlo Simulations

Author: **Luca Ciucci**

Co-Author: **Serena Bruzzesi**

Rust in Scientific HPC for physics simulations

-

Lattice QFT simulations case study

- What is QFT
- Does Rust fit?
- Powers of Rust
- HPC frameworks and libraries
- Integration strategies
- Conclusions
- Bibliography
-



What is QFT

- In a **Field Theory**, the universe is composed of fundamental fields pervading the entire space-time
- The field gives a natural interpretation of probability amplitudes for particles (e.g. the electromagnetic)

$$\begin{cases} x(t) \longrightarrow |\psi\rangle = \psi(x, t) & \text{a state in QM} \\ \Phi(x, t) & \text{a configuration FT} \end{cases}$$

⇓

$$|\Phi(x, t)\rangle \quad \text{QFT}$$

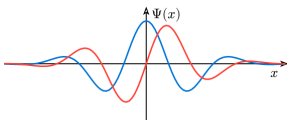


Figure 1: QM in 0 dimension

- Many **interesting properties** does not require to resolve the state of the system:
 - transition amplitudes
 - correlation functions
 - etc.

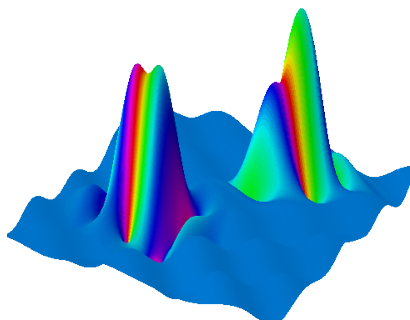


Figure 2: A scalar field configuration,

• What is QFT

- Does Rust fit?
- Powers of Rust
- HPC frameworks and libraries
- Integration strategies
- Conclusions
- Bibliography
-



Connection with Statistical Systems

- Transition amplitudes can be written in terms of a path integral over the field configurations

$$P(\phi)D\phi = e^{iS(\phi)}D\phi$$

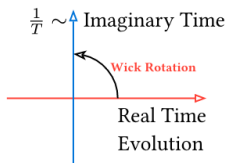


Figure 3: Analytic continuation of the real-time evolution into an imaginary time evolution, related to the **temperature**

- Some quantities can be rewritten in terms of **ground state expectation values** of a statistical system
- One of the approaches is to discretize the space-time in a **lattice**

$$\langle O \rangle \rightarrow E[O] = \frac{\int O(s)e^{-\beta E(s)}}{\int e^{-\beta E(s)}}$$

- What is QFT
 - Does Rust fit?
 - Powers of Rust
 - HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



Monte Carlo Methods

- This integral is hard to compute because of the **curse of dimensionality**

$$\langle O \rangle = \frac{\int O(s) e^{-\beta E(s)} \int e^{-\beta E(s)}$$

- we can use **Monte Carlo methods** to **sample the phase space of the system**
- Markov Chains**: generating samples consists in transitioning across states following a (stochastic) *Markov process*

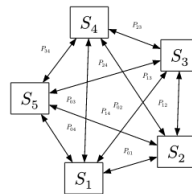


Figure 4: Concept of a Markov Chain

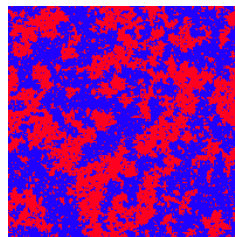


Figure 5: Example of an 2D Ising configuration

- What is QFT
 - Does Rust fit?
 - Powers of Rust
 - HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



Simulation and requirements

- The **simulation** consists in proposing and testing “random” states transitions
- The transitions can be done in parallel (with some care, e.g. when the **action** is **local**)
- Different parts of the grid can be split across different nodes, which requires **synchronization** and **communication** of the **borders**
- The **performance** of the simulation is crucial and can often be offload to **HPC** (High Performance Computing) systems which use modern CPUs and **GPUs**

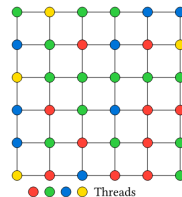


Figure 6: Work on different points of the grid is split across different threads

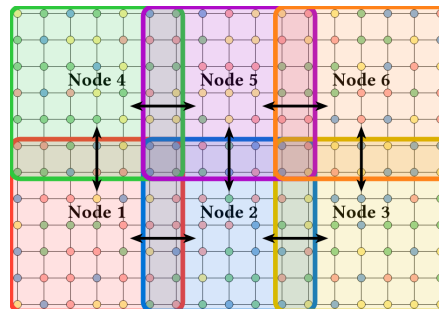


Figure 7: Grid split across multiple nodes

- What is QFT
 - Does Rust fit?
 - Powers of Rust
 - HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



Does Rust fit?

- Simulations should be **efficient**, **fast**, and **predictable**
 - language of choices are often **C** or **Fortran**
 - OpenMP or OpenACC to parallelize the code
- C and Fortran are quite limited, often requiring to “reinvent the wheel” for common tasks, like **serialization**, **RNG**, etc.
- C++ is often avoided because of its **complexity**, **pitfalls**, and **expertise** needed to write efficient code
- Easy to make mistakes in memory unsafe languages, especially with pointers aliasing/no-aliasing situations
- Rust requires the programmer to be very **explicit** on its intents
 - it avoids many of the **pitfalls** of C++ by design
 - many compile time checks to ensure **safety** and **correctness** (no-alias rules, ownership, *Send/Sync*, etc.)
 - no **hidden costs**, clones are explicit, super important for **predictable performance**
- Usually **easy to write and understand**, no need to keep in mind complex implicit rules
- Great **tooling** to help with development and **reproducibility** and **code reusability**

- What is QFT
- **Does Rust fit?**
- Powers of Rust
- HPC frameworks and libraries
- Integration strategies
- Conclusions
- Bibliography
-



Does Rust fit?

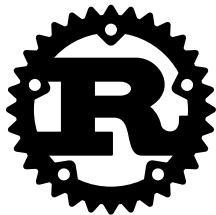
- Simulations should be **efficient**, **fast**, and **predictable**
 - language of choices are often **C** or **Fortran**
 - OpenMP or OpenACC to parallelize the code
- C and Fortran are quite limited, often requiring to “reinvent the wheel” for common tasks, like **serialization**, **RNG**, etc.
- C++ is often avoided because of its **complexity**, **pitfalls**, and **expertise** needed to write efficient code
- Easy to make mistakes in memory unsafe languages, especially with pointers aliasing/no-aliasing situations
- Rust requires the programmer to be very **explicit** on its intents
 - it avoids many of the **pitfalls** of C++ by design
 - many compile time checks to ensure **safety** and **correctness** (no-alias rules, ownership, [Send/Sync](#), etc.)
 - no **hidden costs**, clones are explicit, super important for **predictable performance**
- Usually **easy to write and understand**, no need to keep in mind complex implicit rules
- Great **tooling** to help with development and **reproducibility** and **code reusability**

- What is QFT
- Does Rust fit?
 - Powers of Rust
 - HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



Powers of Rust

Do not spend time on solved problems



- What is QFT
- Does Rust fit?
- Powers of Rust
 - HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



Ease of use

- Researchers and students often need to just dive-in into the project
 - with C offers a lot of freedom but requires a lot of work and **knowledge**
 - diving into a large codebase requires a lot of time to understand the architecture, conventions and methodologies
 - **no clear boundaries** between software components, no clear separation of sync/async code, etc.
 - **programming style** is often not idiomatic and **heterogeneous**
- Easy to express **separation of concerns** in Rust
- splitting code in **crates** and **modules** is easy and natural
- Easy to manage **dependencies**, they usually work out of the box
- Easy to document and navigate
- **Traits** and **generics** offer a natural “formalization” of the concepts in practice

- What is QFT
- Does Rust fit?
- Powers of Rust
 - HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



Ease of use

- Researchers and students often need to just dive-in into the project
 - with C offers a lot of freedom but requires a lot of work and **knowledge**
 - diving into a large codebase requires a lot of time to understand the architecture, conventions and methodologies
 - **no clear boundaries** between software components, no clear separation of sync/async code, etc.
 - **programming style** is often not idiomatic and **heterogeneous**
- Easy to express **separation of concerns** in Rust
- splitting code in **crates** and **modules** is easy and natural
- Easy to manage **dependencies**, they usually work out of the box
- Easy to document and navigate
- **Traits** and **generics** offer a natural “formalization” of the concepts in practice

- What is QFT
- Does Rust fit?
- Powers of Rust
 - HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



Reproducibility and code reusability

- The programmer is encouraged to reuse existing libraries and pin their versions, which is **crucial for reproducibility** in a **scientific context**
- In C a lot of effort is spent on reimplementing common tasks
 - easy to make mistakes and fall into UB traps
- In Rust:
 - practically effortless integrate crates for these tasks
 - reduced system dependencies, which may introduce non reproducible behaviors

- What is QFT
- Does Rust fit?
- Powers of Rust
 - HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



Parallelization and concurrency

- Easy to make subtle mistakes in C, errors caused by **data races** may affect the simulation in unpredictable ways and never be detected
- **Send** and **Sync** traits offers a clear and safe formalization of the concepts with **compile time checks**
- Rust exposes a lot of helpers to **safely share data**, for example `&mut [T]` are not aliased
 - compilers can make a lot of safe assumptions
 - in lattice simulation it is common to see `restrict` everywhere, possibly hiding non-obvious bugs and inconsistencies across compilers if not done with **extreme care**

```

#ifdef __GNUC__
    #define __restrict
#endif

void now(use * __restrict everywhere) {
    // possible hidden UB
}

fn no_worries(data: &mut [T]) {
    // no need for restrict
}

```

- What is QFT
- Does Rust fit?
- Powers of Rust
 - HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



Parallelization and concurrency

- Easy to make subtle mistakes in C, errors caused by **data races** may affect the simulation in unpredictable ways and never be detected
- **Send** and **Sync** traits offers a clear and safe formalization of the concepts with **compile time checks**
- Rust exposes a lot of helpers to **safely share data**, for example `&mut [T]` are not aliased
 - compilers can make a lot of safe assumptions
 - in lattice simulation it is common to see `restrict` everywhere, possibly hiding non-obvious bugs and inconsistencies across compilers if not done with **extreme care**

```

#ifdef __GNUC__
    #define __restrict
#endif

void now(use * __restrict everywhere) {
    // possible hidden UB
}

fn no_worries(data: &mut [T]) {
    // no need for restrict
}

```

- What is QFT
- Does Rust fit?
- Powers of Rust
 - HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



HPC frameworks and libraries

- Established libraries and frameworks in this field include:
 - **Open MPI**
 - **OpenACC**
 - **OpenMP**

- What is QFT
- Does Rust fit?
- Powers of Rust
- **HPC frameworks and libraries**
 - Integration strategies
 - Conclusions
 - Bibliography
 -



Open MPI

- Need for high efficiency **communication** and **synchronization** across **nodes**

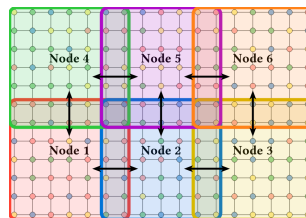


Figure 8: Recall the configuration split across multiple nodes

- Message Passing Interface (**MPI**): a well established **specification** and **library** for parallel programming in **distributed memory systems**.
- Great bindings exist for **Open MPI**: github.com/rsmpi/rsmpi
- Ability to interact with existing MPI code

- What is QFT
- Does Rust fit?
- Powers of Rust
- HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



OpenMP and parallelization

- OpenMP performs great especially when the execution times are constant
- Simulation times are not perfectly constant:
 - accept/reject steps
 - possibly barriers
 - possible interaction with blocking IO
- Some Rust crates offers similar functionality with safe abstractions
 - rayon for data parallelism
 - crossbeam for concurrency and parallelism, advanced GC, etc.

- Often not so difficult to rewrite thread pools, but usually unnecessary

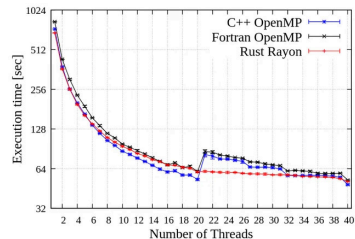


Figure 9: Example benchmark from Eduardo M. Martins et. al. [1]

- What is QFT
- Does Rust fit?
- Powers of Rust
- HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -

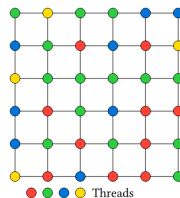


The deal with GPU computing

- GPU computing is hard and also many frameworks to choose from, e.g.:
 - **OpenACC**
 - **CUDA**
 - **OpenCL**
- You cannot write OpenCL and OpenACC code in Rust (but you could interact with bindings)
- GPU computing inherently clashes with some Rust principles

```
fn kernel(conf: &mut [SU3]) {
    // not nice!  ^^^^
}
```

- Many great projects exist and are progressing fast
 - **Rust-GPU** recently rebooted
 - **Rust-CUDA**
 - **wgpu** not HPC tailored but great abstractions
- Rust-GPU and Rust-CUDA are not trivial to setup for beginners, yet (nightly compiler, etc.)



- What is QFT
- Does Rust fit?
- Powers of Rust
- HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



What I'd like to see in the future

Different float types (WIP):

- **f16** and **f80** for experimenting with different precisions and performance trade-offs
 - universal properties are robust under perturbations [2]
 - non-universal properties are always altered
- **f128** not very interesting in lattice simulations, but very useful in other physics simulations (e.g. **astrophysics simulations**)
- More advanced compiler optimizations

```
for neighbor in model.nn(idx) { // <-- easy to make bottlenecks
    // ...
}
```

- What is QFT
- Does Rust fit?
- Powers of Rust
- HPC frameworks and libraries
 - Integration strategies
 - Conclusions
 - Bibliography
 -



Integration strategies

- Great potential for new projects
- C interop offers a simple way to easily integrate with existing codebases
 - great potential to replace common tasks (serialization, **RNG**, debug printing, **state dumping** and **parsing**, etc.)
- Lattice simulation codebases contains very complex concepts, but the codebases are not huge, very feasible to replace single modules at a time
- ⚠ OpenACC code is an hard barrier for Rust: either replace it or use C interop

- What is QFT
- Does Rust fit?
- Powers of Rust
- HPC frameworks and libraries
- [Integration strategies](#)
- Conclusions
- Bibliography
-



Conclusions

- Rust is a great potential for scientific computing, especially in the field of High Performance Computing and Monte Carlo simulations
 - Potential to ease development and dive-in times for new students and researchers
 - Rust integration in existing C projects is very feasible but requires some effort
 - GPU computing is possibly the hardest part, but this will most likely change in the near future
- What is QFT
 - Does Rust fit?
 - Powers of Rust
 - HPC frameworks and libraries
 - Integration strategies
 - [Conclusions](#)
 - Bibliography
 -



Bibliography

- [1] Eduardo M. Martins and Leonardo G. Faé and Renato B. Hoffmann and Lucas S. Bianchessi and Dalvan Griebler, “NPB-Rust: NAS Parallel Benchmarks in Rust.” [Online]. Available: <https://arxiv.org/abs/2502.15536>
- [2] Matteo Bacci, “Stability of Universality Classes under perturbations of Markov Chain Monte Carlo.” [Online]. Available: <https://etd.adm.unipi.it/theses/available/etd-01092025-164459/>

- What is QFT
- Does Rust fit?
- Powers of Rust
- HPC frameworks and libraries
- Integration strategies
- Conclusions
- Bibliography
-