Mostly Painless Scientific Computing with Rust

Srinath Kailasa, Advisor: Timo Betcke

GitHub

Resume

Becoming a Rustacean

Rust has a familiar syntax, inheriting concepts from both object oriented and functional programming paradigms.

Ownership

* Rust's unique memory system enforces that each resource has only one owner at compile time. When the owner goes out of scope, the resource is freed.

* This works in tandem with the 'Borrow Checker', which enforces that there is never more than a single mutable reference to a given resource.

* Together, they allow for **memory errors** to be **caught at compile time**.

Cargo

* Cargo allows for the specification of dependencies via TOML files. Cargo takes responsibility for building and installing all dependencies, except system level libraries.

* Cargo can compile for a range of hardware targets, from x86 to Arm, making deployment often as simple as a single command.

The Alternatives

* Rust is not a competitor to interpreted languages such as Matlab, Julia or Python, and should not aim to be!

* Interpreted languages offer extensions built in compiled languages or use just in time (JIT) compilation to fast machine code for performance sensitive applications so that users familiar with interpreted languages can continue to benefit from the usability of interpreted languages, and enjoy faster performance..

* Developing custom extensions requires developers to maintain complex coupled codebases.

*This tradeoff between language usability and speed is referred to as the **two language problem.** Rust addresses this by offering a fast ergonomic alternative to existing compiled languages.

The Ecosystem

We check the libraries used by Rusty Tree

Distributed Memory Parallelism
rsmpi rsmpi

Multithreading
rayon rayon-rs

GPU Programming
rust-cuda rust-gpu
rust-gpu Embark Studios

Linear Algebra
ndarray-linalg rust-ndarray
BLAS, LAPACK } Blas & Lapack
CBLAS, LAPACKE } in Rust
ndarray-linalg-rs rust-ml

Numerical Data
ndarray rust-ndarray

Foreign Function Interfacing
maturin PyO3

Legend

- Native Rust Library
- Bindings to C++
- Bindings to C
- Bindings to Fortran
- Project Maintainer

Data Visualisation
plotters 38

Data Persistance
hdf5-rust aldanor

Machine Learning & Data Science
linfa rust-ml
Tensorflow Google
ndarray-stats rust-ndarray
... and many more

SIMD
faster Adam Niederer

* The great challenge for the scientific Rust community is standardisation,this list illustrates a small, but popular, fraction of the ecosystem.

* **Many libraries have overlapping concerns**, and many vital libraries are simple bindings for implementations in more established languages, which cannot take advantage of Cargo.

* The scientific Rust ecosystem is rapidly growing. Despite its youth, Rust already supports most functionality needed for high-performance computational science, either natively or via bindings to other languages.

* At UCL we are currently developing a native Rust library to replicate some of the functionality of Eigen. With fast linear algebra routines, and expression templating, built on top of the popular ndarray container crate: github.com/UCL-ARC/householder

Rusty Fast Solvers

Project goals and aims

* We are building a unified software for the fast application of the forward and backward discrete operators that arise from integral equation formulations for elliptic PDEs.

* Our key target application is a **fast solver for electromagnetic scattering**, specified by Maxwell's equations. E.G. from a perfect electric conductor:

$$\begin{aligned} \mathbf{curl}\ \mathbf{curl}\ \mathbf{e} - \kappa^2 \mathbf{e} &= \mathbf{0}, \ \text{in } \Omega^{\pm} \\ \mathbf{e} \times \nu &= \mathbf{0}, \ \text{on } \Gamma \end{aligned}$$

+ Radiation Condition

* A fast solver for Maxwell's equations will allow for the rapid simulations in various domains, from radar systems to material science and medical imaging

* Our goal is for our software to **deploy anywhere**, from a desktop workstation to a supercomputing cluster - to encourage maximum adoption in the community.

Theoretical Background

* We convert PDEs to **Integral Equations** of a general form.

$$a(x)u(x) + b(x) \int_{\Omega} K(x,y)c(y)u(y)dy = f(x)$$

* This reduces the dimensionality of the problem, converting a volume to a surface integral. However, **our discretised matrices are dense.** With $O(N^2)$ storage costs, and naively $O(N^2)$ application and $O(N^3)$ inversion cost

* ***Fast algorithms*** have been developed for the forward [1,2,3] and inverse [4,5] application of the integral operator matrices that arise for certain PDEs in **O(N) at best**.

* There is a gap in research software for the distributed application of operator inverses. Once filled, simulating complex physical systems with multiple right hand sides, from biophysics to electromagnetics would be tractable [6].

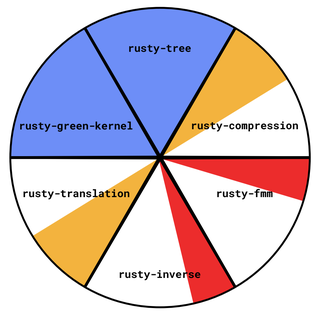

Figure 1 Illustration of progress on the Rusty Fast Solers project. Blue indicates usable libraries, yellow for partially implemented libraries and red for libraries in planning.

Rusty Tree

* Octrees are the foundational data structure for the Fast Multipole Method [1], used to acelerate the application of dense matrices to O(N) in three dimensional simulations.

* Octrees are a spatial decomposition of an three dimensional cube in which the root node encloses the area of interest containing all physical points being meshed, and is recursively partitioned into eight child nodes, until a user defined resolution is reached

* High-performance octree implementations that scale in parallel computing environments are an active area of research [7, 8, 9, 10], with the state of the art implementations focusing on parallel algorithms that apply a Morton encoding (which perserve spatial locality) (fig 3c.) to a given point contained in an octree node, and perform an efficient parallel sort over encoded points. Querying encoded keys for the points they contain is trivial.

* Rusty Tree is an MPI distributed implementation of parallel octrees based on the algorithms first presented in [9,10].

* We perform experiments onthe Rusty system at the Flatiron Institute, with 40 core Skylake nodes.

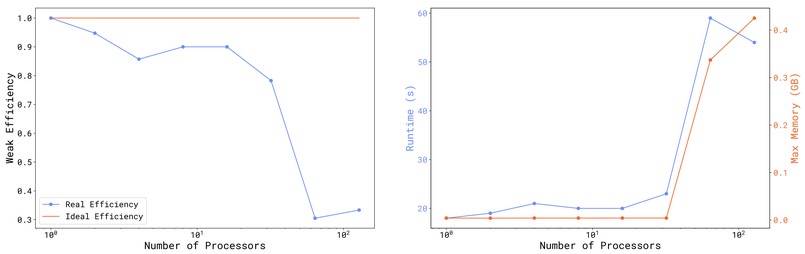

Figure 2 We demonstrate (a) weak scaling efficieny and (b) runtime, for 5e6 randomly dist. points per processor, for a maximum of 640e6 points on 128 processors, used to construct a balanced [9] distributed tree in approximately 55 seconds.

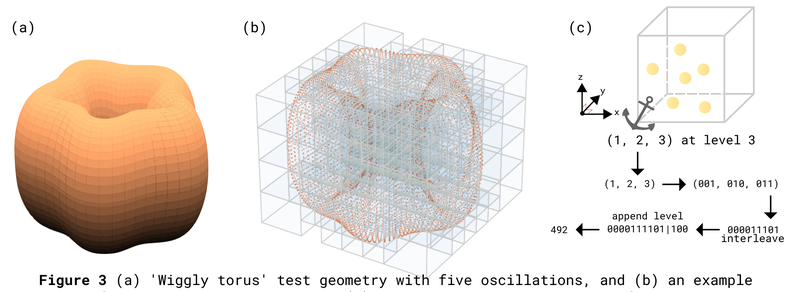

Figure 3 (a) 'Wiggly torus' test geometry with five oscillations, and (b) an example octree for points in this geometry, (c) demonstrates Morton encoding for a node.

References

[1] Greengard, L., & Rokhlin, V. (1987). A fast algorithm for particle simulations. JCP, 73(2), 325-348
[2] Ying, L., Biros, G., & Zorin, D. (2004). A kernel-independent adaptive fast multipole algorithm in two and three dimensions. JCP, 196(2), 591-626
[3] Fong, W., & Darve, E. (2009). The black-box fast multipole method. JCP, 228(23), 8712-8725
[4] Minden, V., Ho, K. L., Damle, A., & Ying, L. (2017). A recursive skeletonization factorization based on strong admissibility. MMS, 15(2), 768-796.
[5] Ambikasaran, S., & Darve, E. (2014). The inverse fast multipole method. arXiv preprint arXiv:1407.1572.
[6] Greengard, L., Gueyffier, D., Martinsson, P. G., & Rokhlin, V. (2009). Fast direct solvers for integral equations in complex three-dimensional domains. Acta Numerica, 18, 243-275.
[7] Burstedde, C., Wilcox, L. C., & Ghattas, O. (2011). p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. SIAM Journal on Scientific Computing, 33(3), 1103-1133.
[8] Malhotra, D., & Biros, G. (2015). PVFMM: A parallel kernel independent FMM for particle and volume potentials. Communications in Computational Physics, 18(3), 808-830.
[9] Sundar, H., Sampath, R. S., & Biros, G. (2008). Bottom-up construction and 2: 1 balance refinement of linear octrees in parallel. SIAM Journal on Scientific Computing, 30(5), 2675-2708.
[10] Sundar, H., Malhotra, D., & Biros, G. (2013). Hyksort: a new variant of hypercube quicksort on distributed memory architectures. In Proceedings of the 27th international ACM conference on international conference on supercomputing (pp. 293-302).

Acknowledgements

* Srinath Kailasa is supported by EPSRC studentship 2417009 and a G-Research PhD Grant.
* Timo Betcke is supported by EPSRC grants EP/W007460/1 & EP/W026260/1