More Usable Flops per Watt: A Story About a Highly Optimised Correlation Function Code

Manodeep Sinha
ASTRO 3D Centre of Excellence, Swinburne

Repo: github.com/manodeep/Corrfunc/
Collaborators: Lehman Garrison
Contributors: Andrew Hearin, Nick Hand
Have you asked these questions?
Have you asked these questions?

- For a list of sources, find all objects that are nearby
Have you asked these questions?

- For a list of sources, find all objects that are nearby
- Compute a correlation function of sources
Have you asked these questions?

- For a list of sources, find all objects that are nearby
- Compute a correlation function of sources
- Calculate all pairwise separations between two sets of points
Have you asked these questions?

- For a list of sources, find all objects that are nearby
- Compute a correlation function of sources
- Calculate all pairwise separations between two sets of points

Then you can use Corrfunc
Why I wrote open-sourced Corrfunc
Why I wrote open-sourced Corrfunc

- Inherited codes took ~5 mins. MCMC would have exceeded the funding duration.
  - fast private version for my specific use-case
Why I wrote open-sourced Corrfunc

- Inherited codes took ~5 mins. MCMC would have exceeded the funding duration.
  - fast private version for my specific use-case

- Created custom code for experts with 6000x speedup (took < 24 hrs to create)
Why I wrote open-sourced Corrfunc

- Inherited codes took ~5 mins. MCMC would have exceeded the funding duration.
  - fast private version for my specific use-case
- Created custom code for experts with 6000x speedup (took < 24 hrs to create)
- Demonstrated the need for a fast, flexible, open-source package
Why I wrote open-sourced Corrfunc

- Inherited codes took ~5 mins. MCMC would have exceeded the funding duration.
  - fast private version for my specific use-case
- Created custom code for experts with 6000x speedup (took < 24 hrs to create)
- Demonstrated the need for a fast, flexible, open-source package
- That initial 5 min calc. now takes ~5 secs with Corrfunc
Code for a Correlation Function

```cpp
for(int i=0;i<N1;i++) {
    for(int j=0;j<N2;j++) {
        double dist = @distance_metric@(point[i], point[j]);
        if(dist < mindist || dist >= maxdist) {
            continue;
        }

        int ibin = @dist_to_bin_index@(dist);
        numpairs[ibin]++;
    }
}
```
for(int i=0;i<N1;i++) {
    for(int j=0;j<N2;j++) {
        double dist = @distance_metric@(point[i], point[j]);
        if(dist < mindist || dist >= maxdist) {
            continue;
        }
        int ibin = @dist_to_bin_index@(dist);
        numpairs[ibin]++;
    }
}
Simple Code is ... simple
Simple Code is … simple

- Ignores domain knowledge

(maxdist << L)

```c
for(int i=0;i<N1;i++) {
    for(int j=0;j<N2;j++) {
        double dist = @distance_metric@(point[i], point[j]);
        if(dist < mindist || dist >= maxdist) {
            continue;
        }

        int ibin = @dist_to_bin_index@(dist);
        numpairs[ibin]++;
    }
}
```
Simple Code is ... simple

- Ignores domain knowledge

\[(\text{maxdist} \ll \text{L})\]

- Not optimal for hardware*

```c
for (int i = 0; i < N1; i++) {
    for (int j = 0; j < N2; j++) {
        double dist = @distance_metric@(point[i], point[j]);
        if (dist < mindist || dist >= maxdist) {
            continue;
        }
        int ibin = @dist_to_bin_index@(dist);
        numpairs[ibin]++;
    }
}
```
Simple Code is ... simple

- Ignores domain knowledge

  (maxdist << L)

- Not optimal for hardware*

- Can not be vectorized by compiler
Why Optimise Code?

H/T to Greg Poole and Jennifer Piscione
Technical Detour
Memory access is slow
Memory access is slow

How fast code runs depends on memory access patterns
Improving/Future-proofing CPU Performance
Improving/Future-proofing CPU Performance

- Only one instruction per clock cycle
  - but, clock speeds have stalled
    - More calculations per clock tick (SIMD/vectorization)
Improving/Future-proofing CPU Performance

- Only one instruction per clock cycle
  - but, clock speeds have stalled
    - More calculations per clock tick (SIMD/vectorization)
- Memory access is slow
  - Layers of (smaller, faster, dedicated) -> (larger, slower, shared) caches
Improving/Future-proofing CPU Performance

- Only one instruction per clock cycle
  - but, clock speeds have stalled
    - More calculations per clock tick (SIMD/vectorization)
- Memory access is slow
  - Layers of (smaller, faster, dedicated) -> (larger, slower, shared) caches
- Power scales as freq^3
  - Multi-cores at lower clock (instead of one core with a 3GHz clock, 2 cores with 2.1 GHz provides 1.4x op/s @ 70% power)
Improving/Future-proofing CPU Performance

- Only one instruction per clock cycle
  - but, clock speeds have stalled
    - More calculations per clock tick (SIMD/vectorization)
- Memory access is slow
  - Layers of (smaller, faster, dedicated) -> (larger, slower, shared) caches
- Power scales as freq^3
  - Multi-cores at lower clock (instead of one core with a 3GHz clock, 2 cores with 2.1 GHz provides 1.4x op/s @ 70% power)

**Vectorized operations with efficient memory access maximises FLOPS/Watt**

ASA ASM, Swinburne, June 2018
Back to Corrfunc
Corrfunc Performance: Single-core

Corrfunc speed-up: $t_{\text{other}}/t_{\text{Corrfunc}}$

- halotools
- kdcount
- Treecorr
- CUTE_box
- scikit-learn KDTree
- SciPy cKDTree
- mlpack RangeSearch

Corrfunc faster
Corrfunc slower

$N_{\text{particles}}$
Corrfunc Performance: Single-core

on github: paper/scripts/generate_code_comparison.py
Corrfunc Performance: Multi-core

![Graph showing Corrfunc speed-up compared to other methods.](image)
Corrfunc Performance: Multi-core

on github: paper/scripts/generate_code_comparison.py
Why Corrfunc is FAST

- Grids extent with cells of $r_{\text{max}}$ (domain knowledge)
- Stores particles contiguously within each cell (memory access)
- Uses vector intrinsics (vectorization)
- Uses OpenMP (multi-core)
Speedup from Vectorization (AVX)
Speedup from Vectorization (AVX)

Vectorization gains for code from the same developer
Framework: PVD calculation reduced from 600+ hours to ~1 min
My proudest moments with Corrfunc

- Found two compiler bugs, one in Intel icc and one in gcc6.4.0
- Bleeding edge kernels cannot be tested because valgrind does not support latest cpus
- Lead Intel HPC engg. starr’ed Corrfunc
- Proud/scared that Corrfunc is now the reference for pair-counting
Conclusions

- **Corrfunc** is optimised using domain knowledge, good memory access pattern, vectorization and OpenMP

- **Corrfunc** is “blazing fast” and
  - modular, user-friendly, documented, tested, OpenMP parallel, flexible API access, ...
  - GPU version coming - thanks to ![ADACS](https://ascl.net/1703.003)

- my highest cited bib-entry for last year ([ascl.net/1703.003](ascl.net/1703.003))
Find Similar Abstracts (with default settings below)
Electronic On-line Article (HTML)
Online Data
Citations to the Article (17) (Citation History)
Refereed Citations to the Article
Associated Articles
Also-Read Articles (Reads History)

Translate This Page

Title: Corrfunc: Blazing fast correlation functions on the CPU
Authors: Sinha, Manodeep; Garrison, Lehman
Publication: Astrophysics Source Code Library, record ascl:1703.003
Publication Date: 03/2017
Origin: ASCL
Keywords: Software
Bibliographic Code: 2017ascl.soft03003S
Conclusions

- **Corrfunc** is optimised using domain knowledge, good memory access pattern, vectorization and OpenMP

- **Corrfunc** is “blazing fast” and
  - modular, user-friendly, documented, tested, OpenMP parallel, flexible API access, …
  - GPU version coming - thanks to ADACS

- my highest cited bib-entry for last year ([ascl.net/1703.003](ascl.net/1703.003))