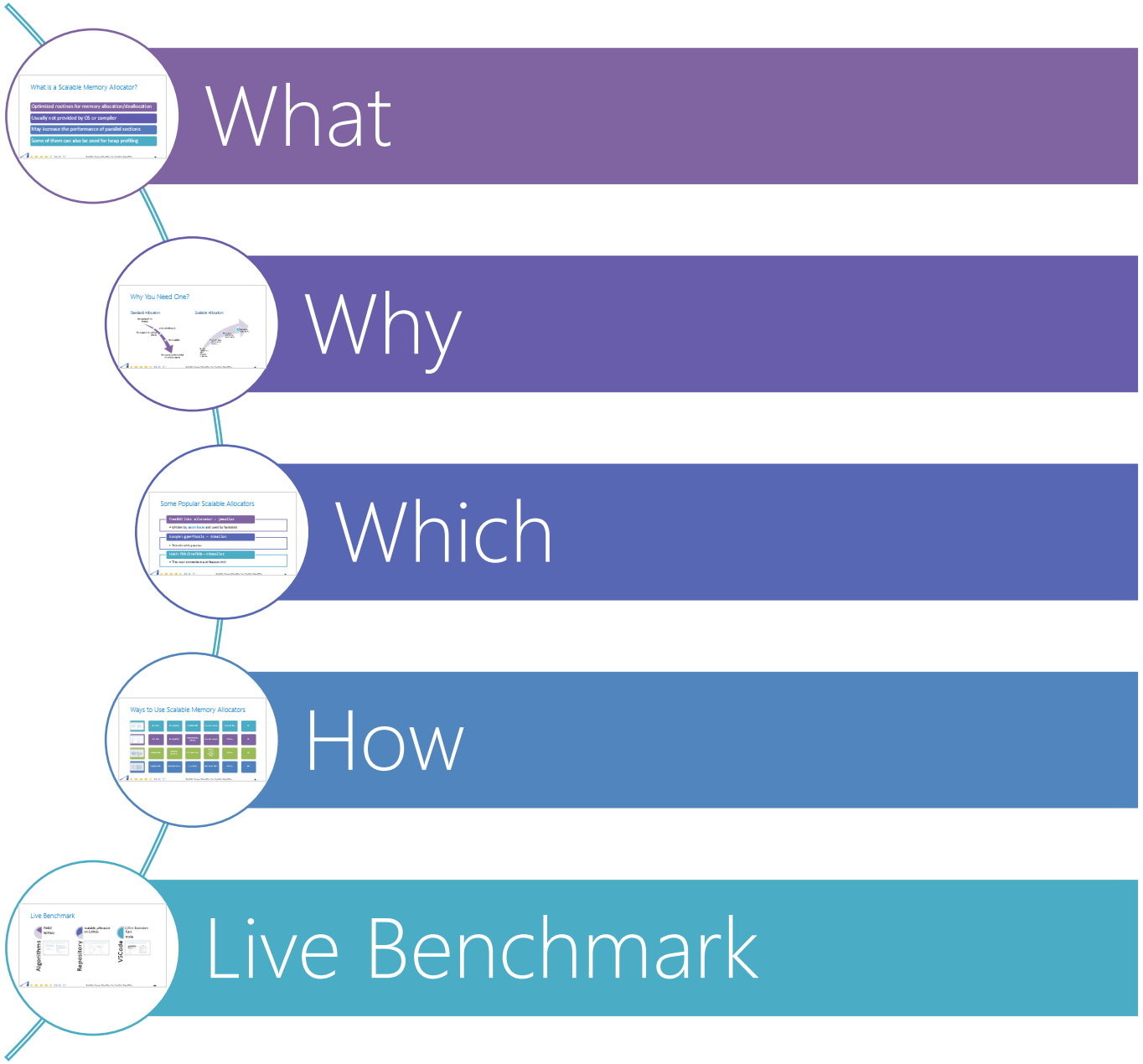
Scalable Memory Allocation for Parallel Algorithms

Armin Sobhani
asobhani@sharcnet.ca
HPC Technical Consultant
SHARCNET | Compute Canada

March 17, 2021

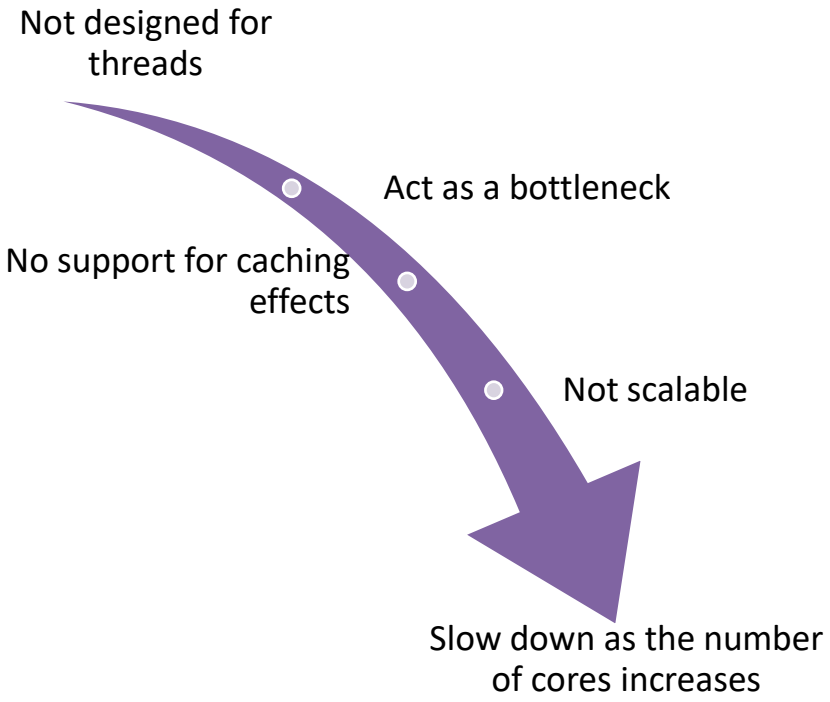


What is a Scalable Memory Allocator?

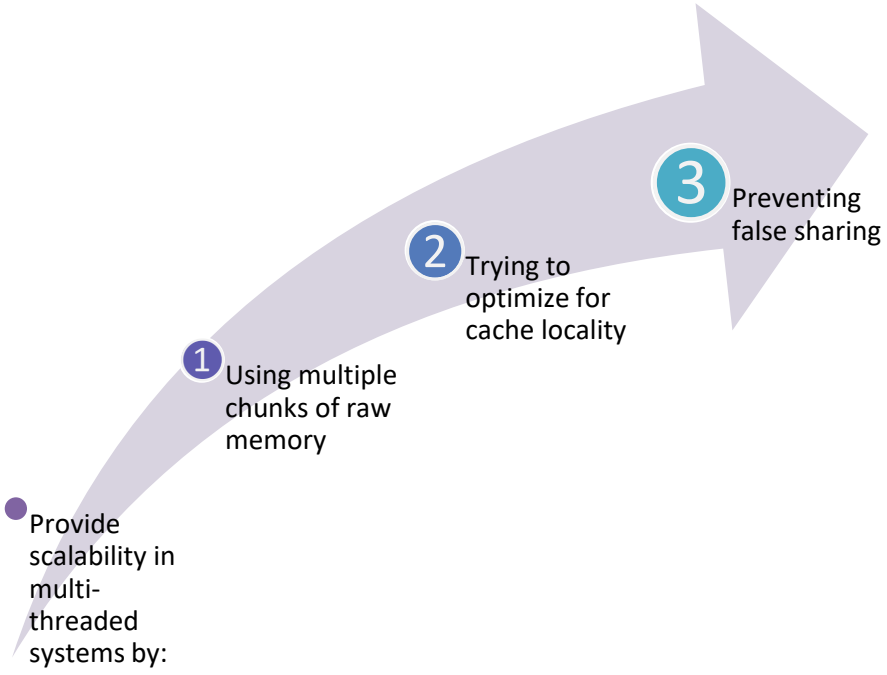
- Optimized routines for memory allocation/deallocation
- Usually not provided by OS or compiler
- May increase the performance of parallel sections
- Some of them can also be used for heap profiling

Why You Need One?

Standard Allocators



Scalable Allocators



Some Popular Scalable Allocators

FreeBSD `libc allocator – jemalloc`

- Written by Jason Evans and used by Facebook

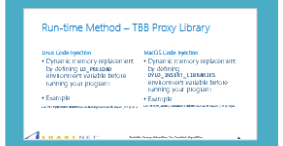
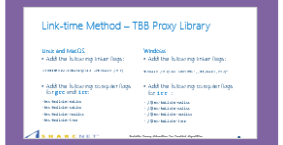
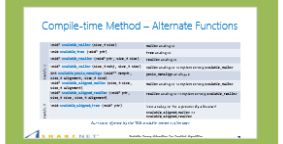
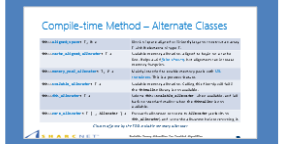
Google's `gperftools – tcmalloc`

- Thread-caching **`malloc`**

Intel's TBB (OneTBB) – `tbbmalloc`

- The most convenient and feature-rich

Ways to Use Scalable Memory Allocators

	Run-time	Proxy Library	LD_PRELOAD	No code changes	Linux and Mac	All
	Link-time	Proxy Library	Static/Dynamic Libraries	No code changes	All Oses	All
	Compile-time	Alternate Functions	C/C++ Functions	`malloc` `calloc` `realloc` `free`	All Oses	All
	Compile-time	Alternate Classes	C++ Classes	`std:allocator`	All Oses	TBB

Run-time Method – TBB Proxy Library

Linux Code Injection

- Dynamic memory replacement by defining **LD_PRELOAD** environment variable before running your program
- Example

```
export LD_PRELOAD=$TBBROOT/lib/intel64/gcc4.8/libtbbmalloc_proxy.so.2
```

MacOS Code Injection

- Dynamic memory replacement by defining **DYLD_INSERT_LIBRARIES** environment variable before running your program
- Example

```
export DYLD_INSERT_LIBRARIES=$TBBROOT/lib/libtbbmalloc_proxy.dylib
```

Run-time Method – TBB Proxy Library

	Linux	MacOS	Windows
Replaceable global C++ operators **`new`** and **`delete`**	Yes	Yes	Yes
Standard C library functions: **`malloc`**, **`calloc`**, **`realloc`**, **`free`**	Yes	Yes	Yes
Standard C library functions (added in C11): **`aligned_alloc`**	Yes		
Standard POSIX* function: **`posix_memalign`**	Yes	Yes	

List of routines replaced by proxy

Link-time Method – TBB Proxy Library

Linux and MacOS

- Add the following linker flags:

```
-L$TBBROOT/lib/intel64/gcc4.8 -ltbbmalloc_proxy
```

- Add the following compiler flags for **`gcc`** and **`icc`**:

```
-fno-builtin-malloc
-fno-builtin-calloc
-fno-builtin-realloc
-fno-builtin-free
```

Windows

- Add the following linker flags:

```
tbbmalloc_proxy.lib /INCLUDE:"__TBB_malloc_proxy"
```

- Add the following compiler flags for **`icc`** :

```
- /Qfno-builtin-malloc
- /Qfno-builtin-calloc
- /Qfno-builtin-realloc
- /Qfno-builtin-free
```

Compile-time Method – Alternate Functions

Family 1	`void* scalable_malloc (size_t size)`	`malloc` analogue
	`void scalable_free (void* ptr)`	`free` analogue
	`void* scalable_realloc (void* ptr, size_t size)`	`realloc` analogue
	`void* scalable_calloc (size_t nobj, size_t size)`	`calloc` analogue complementing `scalable_malloc`
	`int scalable_posix_memalign (void** memptr, size_t alignment, size_t size)`	`posix_memalign` analogue
Family 2	`void* scalable_aligned_malloc (size_t size, size_t alignment)`	`malloc` analogue complementing `scalable_malloc`
	`void* scalable_aligned_realloc (void* ptr, size_t size, size_t alignment)`	`realloc` analogue complementing `scalable_realloc`
	`void scalable_aligned_free (void* ptr)`	free analogue for a previously allocated `scalable_aligned_malloc` or `scalable_aligned_realloc`

Functions offered by the TBB scalable memory allocator

Compile-time Method – Alternate Functions

Family	Allocation Routine	Deallocation Routine	Analogous Library

1	`scalable_malloc`
	`scalable_calloc`
	`scalable_realloc`
		`scalable_free`	C standard library

	`scalable_posix_memalign`		POSIX
2	`scalable_aligned_malloc` `scalable_aligned_realloc`	`scalable_aligned_free`	Microsoft C runtime

Coupling of allocate-deallocate functions by families

Compile-time Method – Alternate Classes

`tbb::aligned_space< T, N >`	Block of space aligned sufficiently large to construct an array **T** with **N** elements of type **T**.
`tbb::cache_aligned_allocator< T >`	Scalable memory allocation, aligned to begin on a cache line. Helps avoid *false sharing*, but alignment can increase memory footprint.
`tbb::memory_pool_allocator< T, P >`	Mainly intended to enable memory pools with STL containers. This is a preview feature.
`tbb::scalable_allocator< T >`	Scalable memory allocation. Calling this directly will fail if the **tbbmalloc** library is not available.
`tbb::tbb_allocator< T >`	Selects **tbb::scalable_allocator** when available, and fall back on standard malloc when the **tbbmalloc** is not available.
`tbb::zero_allocator< T [ , Allocator ] >`	Forwards allocation requests to **Allocator** (defaults to **tbb_allocator**) and zeros the allocation before returning it.

Classes offered by the TBB scalable memory allocator

Compile-time Method – Alternate Classes

std::allocator

```
#include <vector>
#include <algorithm>
#include <execution>


// fill the vector with some data
std::vector<int> v{…};


// sort it in parallel
std::sort(std::execution::par, v.begin(), v.end());
```

tbb::scalable_allocator

```
#include <vector>
#include <algorithm>
#include <execution>

#include <tbb/scalable_allocator.h>

// fill the vector with some data
std::vector<int, tbb::scalable_allocator<int>> v{…};


// sort it in parallel
std::sort(std::execution::par, v.begin(), v.end());
```

Compile-time Method – Alternate Classes

std::allocator

```
#include <vector>
#include <algorithm>
#include <execution>

// fill the vector with some data
std::vector<int> v{…};

// sort it in parallel
std::sort(std::execution::par, v.begin(), v.end());
```

tbb::scalable_allocator

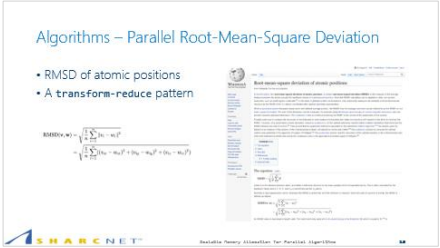
```
#include <vector>
#include <algorithm>
#include <execution>

① #include <tbb/scalable_allocator.h>

// fill the vector with some data
std::vector<int, tbb::scalable_allocator<int>> v{…};

② // sort it in parallel
std::sort(std::execution::par, v.begin(), v.end());
```

Live Benchmark

RMSD

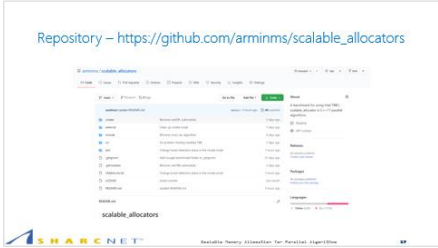
NORM2

Algorithms



scalable_allocators on GitHub

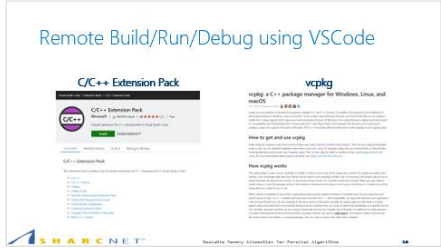
Repository



C/C++ Extension Pack

vcpkg

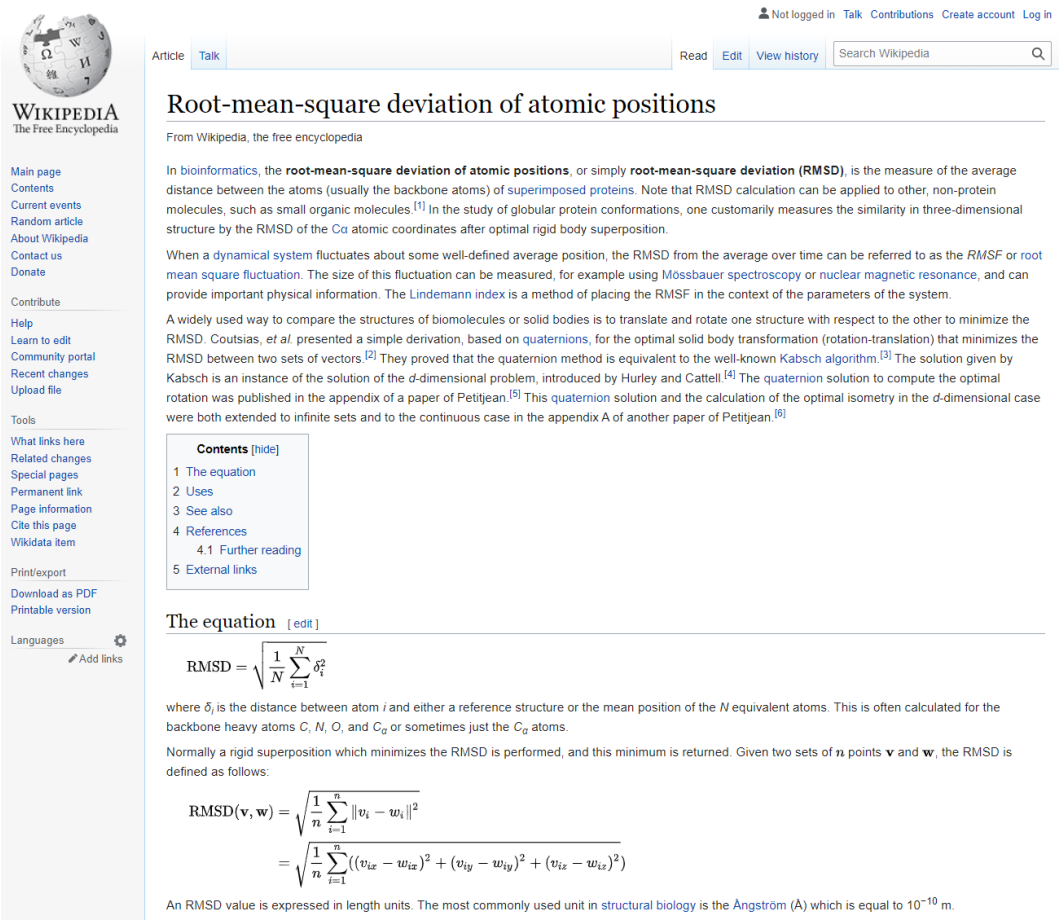
VSCode



Algorithms – Parallel Root-Mean-Square Deviation

- RMSD of atomic positions
- A `transform-reduce` pattern

$$\mathrm{RMSD}(\mathbf{v},\mathbf{w}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\|v_i - w_i\|^2}$$
$$= \sqrt{\frac{1}{n}\sum_{i=1}^{n}((v_{ix}-w_{ix})^2 + (v_{iy}-w_{iy})^2 + (v_{iz}-w_{iz})^2)}$$



Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

Languages
Add links

Root-mean-square deviation of atomic positions

From Wikipedia, the free encyclopedia

In bioinformatics, the **root-mean-square deviation of atomic positions**, or simply **root-mean-square deviation (RMSD)**, is the measure of the average distance between the atoms (usually the backbone atoms) of superimposed proteins. Note that RMSD calculation can be applied to other, non-protein molecules, such as small organic molecules.[1] In the study of globular protein conformations, one customarily measures the similarity in three-dimensional structure by the RMSD of the Cα atomic coordinates after optimal rigid body superposition.

When a dynamical system fluctuates about some well-defined average position, the RMSD from the average over time can be referred to as the *RMSF* or root mean square fluctuation. The size of this fluctuation can be measured, for example using Mössbauer spectroscopy or nuclear magnetic resonance, and can provide important physical information. The Lindemann index is a method of placing the RMSF in the context of the parameters of the system.

A widely used way to compare the structures of biomolecules or solid bodies is to translate and rotate one structure with respect to the other to minimize the RMSD. Coutsias, *et al.* presented a simple derivation, based on quaternions, for the optimal solid body transformation (rotation-translation) that minimizes the RMSD between two sets of vectors.[2] They proved that the quaternion method is equivalent to the well-known Kabsch algorithm.[3] The solution given by Kabsch is an instance of the solution of the *d*-dimensional problem, introduced by Hurley and Cattell.[4] The quaternion solution to compute the optimal rotation was published in the appendix of a paper of Petitjean.[5] This quaternion solution and the calculation of the optimal isometry in the *d*-dimensional case were both extended to infinite sets and to the continuous case in the appendix A of another paper of Petitjean.[6]

Contents [hide]
1 The equation
2 Uses
3 See also
4 References
4.1 Further reading
5 External links

The equation [edit]

$$\mathrm{RMSD} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\delta_i^2}$$

where δ_i is the distance between atom *i* and either a reference structure or the mean position of the *N* equivalent atoms. This is often calculated for the backbone heavy atoms *C*, *N*, *O*, and C_α or sometimes just the C_α atoms.

Normally a rigid superposition which minimizes the RMSD is performed, and this minimum is returned. Given two sets of n points $\mathbf{v}$ and $\mathbf{w}$, the RMSD is defined as follows:

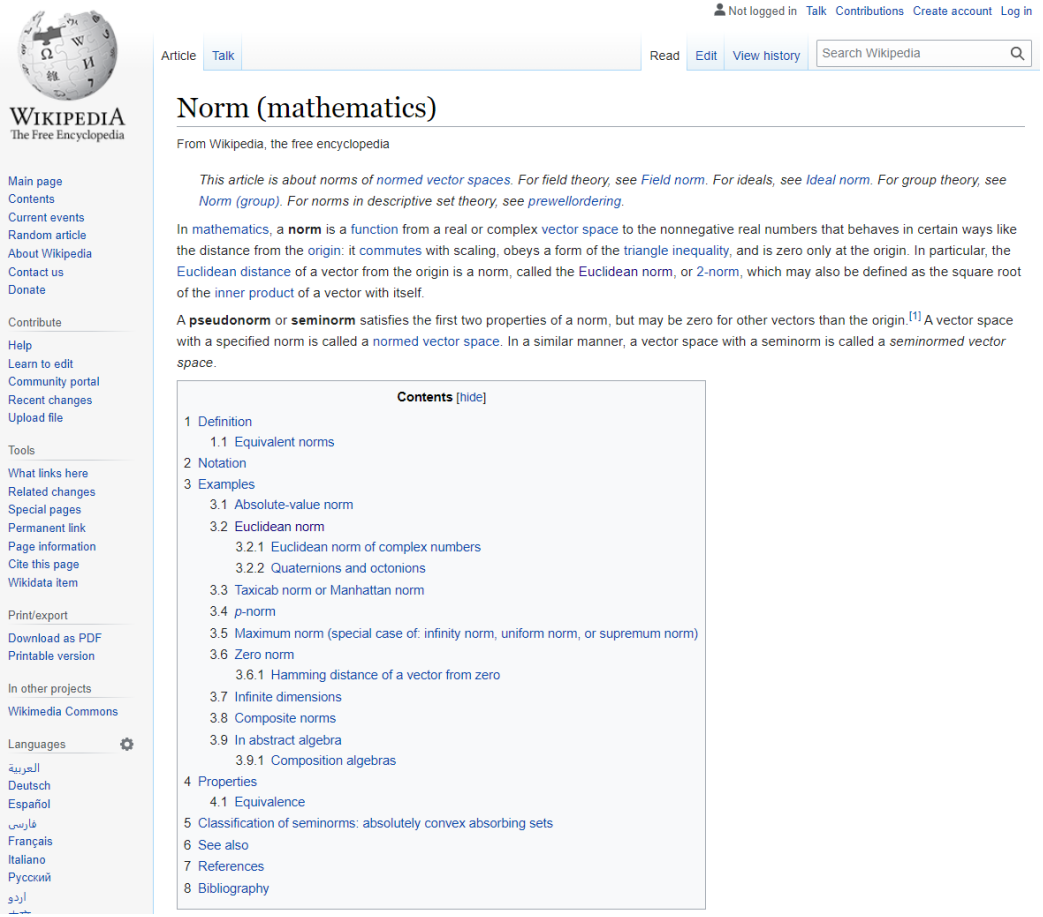
$$\mathrm{RMSD}(\mathbf{v},\mathbf{w}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\|v_i - w_i\|^2}$$
$$= \sqrt{\frac{1}{n}\sum_{i=1}^{n}((v_{ix}-w_{ix})^2 + (v_{iy}-w_{iy})^2 + (v_{iz}-w_{iz})^2)}$$

An RMSD value is expressed in length units. The most commonly used unit in structural biology is the Ångström (Å) which is equal to 10^{-10} m.

Algorithms – Parallel Euclidean Norm

- AKA `norm2` or `2-norm`
- A `reduce` pattern

$$\|\boldsymbol{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}.$$



Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

In other projects
Wikimedia Commons

Languages
العربية
Deutsch
Español
فارسی
Français
Italiano
Русский
اردو

Norm (mathematics)

From Wikipedia, the free encyclopedia

This article is about norms of normed vector spaces. For field theory, see Field norm. For ideals, see Ideal norm. For group theory, see Norm (group). For norms in descriptive set theory, see prewellordering.

In mathematics, a **norm** is a function from a real or complex vector space to the nonnegative real numbers that behaves in certain ways like the distance from the origin: it commutes with scaling, obeys a form of the triangle inequality, and is zero only at the origin. In particular, the Euclidean distance of a vector from the origin is a norm, called the Euclidean norm, or 2-norm, which may also be defined as the square root of the inner product of a vector with itself.

A **pseudonorm** or **seminorm** satisfies the first two properties of a norm, but may be zero for other vectors than the origin.[1] A vector space with a specified norm is called a normed vector space. In a similar manner, a vector space with a seminorm is called a *seminormed vector space*.

Contents [hide]

1 Definition
1.1 Equivalent norms
2 Notation
3 Examples
3.1 Absolute-value norm
3.2 Euclidean norm
3.2.1 Euclidean norm of complex numbers
3.2.2 Quaternions and octonions
3.3 Taxicab norm or Manhattan norm
3.4 *p*-norm
3.5 Maximum norm (special case of: infinity norm, uniform norm, or supremum norm)
3.6 Zero norm
3.6.1 Hamming distance of a vector from zero
3.7 Infinite dimensions
3.8 Composite norms
3.9 In abstract algebra
3.9.1 Composition algebras
4 Properties
4.1 Equivalence
5 Classification of seminorms: absolutely convex absorbing sets
6 See also
7 References
8 Bibliography

Repository – https://github.com/arminms/scalable_allocators

arminms / scalable_allocators

Unwatch 1 | Star 0 | Fork 0

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

main | 1 branch | 0 tags | Go to file | Add file | Code

asobhani Update README.md 8882013 5 hours ago 48 commits

cmake	Remove oneDPL submodule	3 days ago
external	Clean up cmake script	3 days ago
include	Remove rmsd_vec algorithm	4 days ago
src	Fix problem finding installed TBB	3 days ago
test	Change boost detection place in the cmake script	7 hours ago
.gitignore	Add Google benchmark folder to .gitignore	25 days ago
.gitmodules	Remove oneTBB submodule	3 days ago
CMakeLists.txt	Change boost detection place in the cmake script	7 hours ago
LICENSE	Initial commit	last month
README.md	Update README.md	5 hours ago

README.md

scalable_allocators

About

A benchmark for using Intel TBB's scalable_allocator in C++17 parallel algorithms

Readme

MIT License

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

CMake 62.8% C++ 37.2%

Remote Build/Run/Debug using VSCode

C/C++ Extension Pack

Visual Studio Code > Extension Packs > C/C++ Extension Pack

C/C++ Extension Pack

Microsoft | 348,868 installs | ★★★★★ (1) | Free

Popular extensions for C++ development in Visual Studio Code.

Install Trouble Installing?

Overview Version History Q & A Rating & Review

C/C++ Extension Pack

This extension pack includes a set of popular extensions for C++ development in Visual Studio Code:

- C/C++
- C/C++ Themes
- CMake
- CMake Tools
- Remote Development Extension Pack
- GitHub Pull Requests and Issues
- Visual Studio Codespaces
- LiveShare Extension Pack
- Doxygen Documentation Generator
- Better C++ Syntax

vcpkg

vcpkg: a C++ package manager for Windows, Linux, and macOS

12/11/2020 • 4 minutes to read •

vcpkg is a cross-platform command-line package manager for C and C++ libraries. It simplifies the acquisition and installation of third-party libraries on Windows, Linux, and macOS. If your project uses third-party libraries, we recommend that you use vcpkg to install them. vcpkg supports both open-source and proprietary libraries. All libraries in the vcpkg Windows catalog have been tested for compatibility with Visual Studio 2015, Visual Studio 2017, and Visual Studio 2019. Between the Windows and Linux/macOS catalogs, vcpkg now supports thousands of libraries. The C++ community adds more libraries to both catalogs on an ongoing basis.

How to get and use vcpkg

Install vcpkg by making a local clone from its GitHub repo https://github.com/Microsoft/vcpkg. Then run the vcpkg-bootstrapper script to set it up. For detailed installation instructions, see Install vcpkg. To integrate vcpkg with your Visual Studio or Visual Studio Code development environment, see Integrate vcpkg. Then, to use vcpkg to install or update a library, see Manage libraries with vcpkg. For more information about vcpkg commands, see vcpkg command-line reference.

How vcpkg works

The vcpkg project is open-source, available on GitHub. A *clone* or local copy of the vcpkg repo contains the vcpkg executable and a *catalog*, a list of packages that describe a library and its options. Each package includes one or more *ports*, information about how to obtain and build the library from sources, or download a binary version, for a specific target environment. When you use vcpkg to install a library, it uses the package and port information to download and produce a local copy of the library in a subdirectory of the vcpkg directory, ready for you to use.

When a library is available in source form, vcpkg downloads sources instead of binaries. It compiles those sources using the most recent version of the C or C++ compiler and tools that it can find. For C++ ABI compatibility, it's important that both your application code and any libraries you use are compiled by the same version of the same compiler. By using vcpkg, you eliminate or at least greatly reduce the potential for mismatched binaries and the problems they can cause. In teams that standardize on a specific version of a compiler, one team member can use vcpkg to download sources and compile a set of binaries. It's inefficient to make everyone on a team download and build common libraries. One team member can use the `vcpkg export` command to create a common zip file of the binaries and headers, or a NuGet package. Then, it's easy to share it with other team members.

Installing Dependencies with **vcpkg**

Without unit tests

```
$ ./vcpkg install tbb benchmark
```

With unit tests

```
$ ./vcpkg install tbb benchmark boost-system boost-test
```