Documentation

`solidfmm`

Version 1.5

Matthias Kirchhart

May 2023

Dr. Matthias Kirchhart

E-mail: kirchhart@acom.rwth-aachen.de
Website: https://www.linkedin.com/in/matthiaskirchhart

Contents

Part I

User Guide

Chapter 1

Quick Start

This chapter is written for people who already have an understanding of the fast multipole method and the expansions it uses. The definition of these expansions alongside with the notational conventions employed by `solidfmm` are given Chapter 2. That chapter also contains an introduction to these topics. A complete reference of the user interface is given in Chapter 3.

We assume that you are familiar with the command line and a POSIX shell, that you have some background in programming in C++, and that you know how to use the standard development toolchain on GNU/Linux systems.

1.1 Licencing and Copyright

I, Matthias Kirchhart, am the sole author and copyright holder of `solidfmm`. `solidfmm` is free software. You may use, modify, and redistribute `solidfmm` under the terms and conditions of the 'GNU General Public License' (GPL) as published by the Free Software Foundation, either version three, or, at your option, any later version. You should have received a copy of this licence document together with `solidfmm`, it can also be found in Appendix B of this book.

The GPL gives you a lot of freedoms. However, it is a copyleft licence. Consider you write your own piece of software that makes use of `solidfmm`. Then, in case you decide to publish or redistribute your software, you *must* distribute it under the terms of the GPL. In particular, the GPL *does not* allow you to incorporate `solidfmm` into closed-source, non-free, commercial software for retail. If you wish to do so, please contact me to negotiate alternative licencing options.

1.2 Installing `solidfmm`

1.2.1 System Requirements

To compile and install `solidfmm` you will need a fairly recent version of the GNU Compiler Collection (`g++`) *or* LLVM (`clang++`) with support for C++20.

The only external library dependency is `hwloc`,[1] which is necessary for the multi-threading functionality. Apart from this, on Unix systems you will only need standard development tools, i. e., a POSIX compatible shell (like `bash`), and `make`. It should also be possible to compile under Windows using Cygwin or MinGW, but I have not tested this. If you downloaded `solidfmm` by cloning the GitHub repository, you will additionally need the GNU Autotools: `autoconf`, `automake`, `libtool`, as well as `pkg-config` and the software they depend on.

The reason for requiring specific compilers is the following. The performance critical parts of the code—the so-called microkernels—are written in assembly language, in the form of *inline assembly* which is directly embedded in the C++ code. Unfortunately, there is no consensus on how to do this, and different compilers have different formats. Even worse, Microsoft's compiler (MSVC) simply does not support inline assembly at all anymore.[1] `solidfmm` uses GCC's syntax, because GCC is available for almost all platforms, it is free software, and fairly up-to-date with current C++ standards. LLVM also uses the same syntax. Additionally, on x86 platforms the code uses GCC's feature `__builtin_cpu_supports()` to detect the availability of AVX and AVX-512 at runtime. This extension is also supported by LLVM.

1.2.2 Download Sources

`solidfmm` is available from GitHub at `https://github.com/vorticle/solidfmm` in two forms

1. *Recommended:* As so-called 'tarballs', i. e., compressed archives of the source code which come with the readily prepared configuration scripts. Additionally, digital signatures are provided to validate the authenticity of the code. You can find these releases under the 'Releases' section on the GitHub project page, a little bit hidden on the right of the page.

2. As a source repository which can be cloned using the `git` versioning tool. This repository does not contain the prepared configuration scripts; you will additionally need the GNU Autotools to generate these scripts from the source code.

We highly recommend the first option, as it is simpler to use and comes with the additional security benefit of digital signatures.

1.2.3 Verification (Optional)

The next step is to verify the authenticity of the downloaded files. For this you will need the GNU Privacy Guard, also known as GnuPG.[2] Its executable is commonly called `gpg`. It comes preinstalled by default on most current GNU/Linux distributions. Here we will only describe how to use this software to verify the authenticity of `solidfmm` on GNU/Linux systems, a detailed user guide of `gpg` is given on its project website. We will only describe the process for the tarball release, which is the recommended download option. For verifying a cloned GitHub repository we assume that you have sufficient familiarity with `git` to perform the verification yourself.

[1]We quote: 'Inline assembly is not supported on the ARM and x64 processors.' `https://docs.microsoft.com/en-us/cpp/assembler/inline/inline-assembler?view=msvc-170`

Setting up GnuPG

GnuPG only needs to be set up once. If you have already done so during installations of previous versions, you can safely skip this section.

You will need your own keypair. If you do not know what this is, you probably do not have one yet. In this case you can easily create one by entering

```
gpg --gen-key
```

and following the instructions on your screen. This a local procedure: no data is submitted to the internet and your privacy is preserved.

Once you have your own keypair, you can import my key into GnuPG. My public key is included in the repository, it can be downloaded individually from GitHub.

```
gpg --import kirchhart_pubkey.gpg
```

You now need to double check if my key itself has not been compromised. This can *only* be achieved by *manually* comparing it to a second, trustworthy source. To do this, first display the key's fingerprint:

```
gpg --fingerprint "Matthias Kirchhart"
```

This command should show something like:

```
pub   rsa4096 2012-11-02 [SC]
      BC55 D501 A8DC 2D9B D2C9  95F3 00D1 76A4 818C 2BCE
uid           [ ultimativ ] Matthias Kirchhart <matthias.kirchhart@rwth-aachen.de>
uid           [ ultimativ ] Matthias Kirchhart <kirchhart@acom.rwth-aachen.de>
sub   rsa4096 2012-11-02 [A]
sub   rsa4096 2012-11-02 [E]
```

The long string of characters 'BC55 D501 A8DC…' is my public key's fingerprint. You have to compare this fingerprint with the one given in the 'About' section on my LinkedIn page: https://www.linkedin.com/in/matthiaskirchhart/. *Only if the two fingerprints match **completely***, you have successfully imported my public key. If they do not match, something strange is going on and you should abort immediately, check your computer for viruses, etc.

Usually, however, things should work out just fine. In this case, you sign this process off by adding your signature:

```
gpg --sign-key "Matthias Kirchhart"
```

This procedure might seem rather tedious, but security comes at a price. Also note that these steps only need to be carried out once. You will never need to do this again, unless you move to a different computer and forget moving your GnuPG data. You do *not* need to set up GnuPG again when installing updates to `solidfmm`.

Verifying the Tarball

Once you set up GnuPG, verifying the tarball is easy. Simply run the following command in the folder into which you downloaded the tarball and the signature file.

```
gpg --verify solidfmm-1.5.tar.xz.asc solidfmm-1.5.tar.xz
```

The output should contain a message stating that the signature is correct. If so, verification is complete. Otherwise you have downloaded compromised files and should abort *immediately*. Replace the version number 1.5 with the version you downloaded in the above command, if necessary.

1.2.4 Unpacking the Tarball

Once you have downloaded and (optionally) verified the tarball, it can be unpacked as follows.

```
tar Jxf solidfmm-1.5.tar.xz
```

For this command to work, you will need `tar`[3] and the XZ utils,[4] which come preinstalled by default on most current GNU/Linux systems. This should create a folder named `solidfmm-1.5`, containing the source code of `solidfmm`. Enter this folder by typing

```
cd solidfmm-1.5
```

1.2.5 Preparing a Cloned Repository

If you instead downloaded `solidfmm` from GitHub by cloning the repository using the `git` versioning tool, you will need the GNU Autotools to generate the build system and the necessary scripts.

```
autoreconf --install
```

After this the compilation and installation instructions in the next subsection also apply to the cloned repository.

1.2.6 Configuring and Compiling

solidfmm uses the GNU Autotools to configure, build, and install the library. These tools give you great flexibility in doing this and are well beyond the scope of this manual. These tools are, for example, described in detail in John Calcote's book.[5] In the simplest setting you can just enter the source directory and configure and compile using:

```
./configure && make
```

The configure script allows you to control many aspects of the process. For example, it is possible to only build the static library, or to only compile the dynamically linked version. You can chose the compiler, linker, optimisation flags, installation directories, and many more using environment variables and options to the script. It is even possible to cross-compile for a different system. If you want to install solidfmm in a non-standard location, you can do so by passing the --prefix option:

```
./configure --prefix=/your/preferred/location && make
```

However, you should only use this option if you know what you are doing: if you install solidfmm to a non-standard location, you will need to configure your compiler and linker such that they know where to look for solidfmm when using the library. A quick overview over some of the options that configure supports can be seen by executing the following command:

```
./configure --help
```

More detailed options are given in the file INSTALL.

One important aspect for solidfmm is the choice of compiler. Only recent versions of the GNU Compiler Collection, i. e., g++,[6] and LLVM, i. e., clang++,[7] are supported. To specify the compiler to use, you can use the environment variable CXX. For example:

```
# To compile using clang++ use:
CXX=clang++ ./configure && make

# To compile using g++ use:
CXX=g++ ./configure && make
```

If you do not specify CXX, your system's default compiler will be used.

1.2.7 Installing

When compilation is complete, you can install solidfmm. Unless you specified a different --prefix when running configure, solidfmm will be installed into your system's default locations for libraries and headers. These directories are protected, and can only be written to by the system administrator. On Unix systems you would thus first change to the administrator's account and install afterwards:

```
su # Changes to the system administrator's account, requires his or her password
make install
exit # Important: leave administrator's account afterwards
```

Some current GNU/Linux distributions do not allow the use of `su` anymore, because they claim it would be unsafe. As so often, when used properly, this is not true; however `su` does have a history of being used incorrectly. For this reason these distributions promote the use of `sudo` instead. If the above command does not work, on these systems, you can try using the following instead:

```
sudo make install
```

This command requires *your* password and not that of the administrator. It will only work if the system administrator has granted you the necessary rights.

1.3 Using the Library

1.3.1 Headers and Linker Flags

The user interface consists of only four header files. Simply `#include` them at the beginning of your source files:

```
#include <solidfmm/solid.hpp>
#include <solidfmm/handles.hpp>
#include <solidfmm/harmonics.hpp>
#include <solidfmm/translations.hpp>
```

All classes and functions of the library lie in the C++ namespace `solidfmm`. When creating your programme, you will need to link against the library. For example, when using `g++`, this can be achieved by adding the flag `-lsolidfmm` as follows:

```
# Compile your programme
g++ -o yourprogramme.o -c yourprogramme.cpp

# Link it into an executable
g++ -o yourprogramme -lsolidfmm yourprogramme.o
```

Since version 1.5, `solidfmm` also supports the `pkg-config` tool to automatically discover the right linkage and preprocessor flags. If you have this tools installed on your system, you can use it to query the correct flags:

```
pkg-config --libs   solidfmm   # Prints Flags for linking against solidfmm
pkg-config --cflags solidfmm   # Flags for the C-preprocessor to find headers
```

1.3.2 Creating Expansions

Suppose we were given N point charges at locations $(x_i, y_i, z_i)^\top \in \mathbb{R}^3$ with associated charges $m_i \in \mathbb{R}$, $i = 0, \dots, N-1$. We now seek to create a multipole expansion of order $P = 15$ around the point $(x_A, y_A, z_A)^\top \in \mathbb{R}^3$. To do this so-called P2M operation in `solidfmm`, one could use the following code.

```
using solidfmm::solid;
using solidfmm::fmadd;
using solidfmm::harmonics::R; // Regular harmonics.

size_t P { 15 };              // The order of the expansion you want to use.
size_t N;                     // Number of point charges, initialised somewhere else.
const double *x, *y, *z, *m; // Coordinates and charges, initialised somewhere else.

// Objects of type solid store coefficients of multipole or local expansions.
// Pass the desired order of the expansion. Initialised with zero coefficients.
// You can also use "float" for single precision.
solid<double> M( P );

// Add the contribution of each point mass to multipole expansion.
for ( size_t i = 0; i < N; ++i )
    fmadd( m[i], R<double>(P,x[i]-xA,y[i]-yA,z[i]-zA), M );
```

After this, the object `M` of type `solid<double>` contains the coefficients $M_n^m \in \mathbb{C}, n = 0, \dots, P-1,\ m = -n, \dots, n$ of the multipole expansion.

Although much less common, local expansions can be created directly from the given point charges (P2L) in almost the exact same way. All we need to do is to replace the regular harmonics with the singular ones. Thus, assume we wanted to create a local expansion for the same charges around the centre $(x_B, y_B, z_B)^\top \in \mathbb{R}^3$. This can be achieved as follows:

```
using solidfmm::solid;
using solidfmm::fmadd;
using solidfmm::harmonics::S; // Singular harmonics.

solid<double> L( P );
for ( size_t i = 0; i < N; ++i )
    fmadd( m[i], S<double>(P,x[i]-xB,y[i]-yB,z[i]-zB), L );
```

We note that `solidfmm` also supports vector-valued expansions. For this we refer the reader to Chapter 3.

1.3.3 Evaluating Expansions

Evaluation of multipole and local expansions can be done using `dot`. Assume we were given the coefficients $M_n^m \in \mathbb{C}, n = 0, \dots, P-1, m = -n, \dots, n$ of a multipole expansion of order P around the centre $(x_A, y_A, z_A)^\top \in \mathbb{R}^3$. To evaluate this expansion at some point $(x, y, z)^\top \in \mathbb{R}^3$, you would do the following:

```
using solidfmm::dot;
using solidfmm::solid;
using solidfmm::harmonics::S; // Singular harmonics.

solid<double> M;    // Given coefficients,     initialised somewhere else.
double xA, yA, zA;  // Given expansion centre, initialised somewhere else.
double x,  y,  z;   // Given evaluation point, initialised somewhere else.

double result { 0 };
dot( M, S<double>(P,x-xA,y-yA,z-zA), &result );
```

Local expansions work similarly: here you need to replace the singular harmonics with the regular ones. Thus, assuming you were given the coefficients $L_n^m \in \mathbb{C}$, $n = 0, \dots, P-1$, $m = -n, \dots, n$ of a local expansion at centre $(x_B, y_B, z_B)^\top \in \mathbb{R}^3$, you would evaluate it as follows:

```
using solidfmm::dot;
using solidfmm::solid;
using solidfmm::harmonics::R; // Regular harmonics.

solid<double> L;    // Given coefficients,     initialised somewhere else.
double xB, yB, zB;  // Given expansion centre, initialised somewhere else.
double x,  y,  z;   // Given evaluation point, initialised somewhere else.

double result { 0 };
dot( L, R<double>(P,x-xB,y-yB,z-zB), &result );
```

Again, we remark that `solidfmm` also supports vector-valued expansions and refer to Chapter 3 for this.

1.3.4 Translations

We now come to the main reason for the existence of this library: to provide an efficient implementation of the translation operators M2M, M2L, and L2L. In this section we describe how to use the `multithreaded_scheduler`, as it is most easy to use. If you prefer to do the workbalancing and scheduling of threads yourself, you can use the single threaded functions as discussed in Section 3.5.

In `solidfmm`, a translation is described using a `translation_info` object:

```
template <typename real>
struct translation_info
{
    const solid<real> *source;  // Source coefficients
          solid<real> *target;  // Target coefficients

    real x, y, z;               // Shift vector.
};

extern template class translation_info<float>;
extern template class translation_info<double>;
```

The result of translating the `source` solid is *added* to `target`, making accumulation possible. So in pseudo-code we have:

$$\texttt{target} \quad \texttt{+=} \quad \text{translate}(\texttt{source}, x, y, z);$$

where 'translate' can mean any of M2M, M2L, or L2L. Note that `solidfmm` supports translations of mixed orders, so it is important that you initialise the `target` to the correct order. Also note that vector-valued expansions are also supported. Default-constructed objects of type `solid` are of order zero and dimension 1; so in this case nothing happens. If in doubt, you can explicitly initialise the targets using the `resize()` member function. For example, you could use the following code:

```
// If in doubt if your targets were already initialised to the desired
// output order, you can use the following code.

// Large list of translations, initialised somewhere else
std::vector< translation_info<double> > translations;

for ( auto i = translations.begin(); i != translations.end(); ++i )
{
        i->target->resize(P); // Choose your desired output order here.
        i->target->zeros();   // Initialise target to zero.
}
```

The easiest way to carry out translations is the `multithreaded_translator` class. By default, an instance of this class will use *all available cores* to carry out as many translations in parallel as possible. It takes care of data races, so accumulation will work as expected. However, it does not do dependency checking, so a list of translations where some `solid`s are *both* a source and a target will lead to undefined behaviour. In particular, in the usual workflow of the fast multipole method, it usually works to pass a list of *all* M2L translations; however M2M and L2L translations must be carried out level-wise.

```
// Large list of translations, initialised somewhere else
std::vector< translation_info<double> > translations;
```

```
// Maximum order P that we are going to use, initialised somewhere else.
size_t P;

// Expensive to create! Only create once at program start-up if possible.
// Will by default use all available cores on the system and distribute work
// equally among them.
multithreaded_translator<double> T( P );

// On systems with different kinds of cores, e.g., performance cores
// and efficiency cores, it makes sense to measure the throughput and schedule
// work accordingly. Uncomment the following line for this. Measurement is
// expensive, it should be done only once at program start-up.
//T.schedule_by_measuring_throughput();

// Carry out, e.g., m2l translations. m2m and l2l are called analogously.
T.m2l( translations.data(), translations.data() + translations.size() );
```

By default, the translator class will carry out a check at runtime if the list of translations is valid: does no source or target exceed order P? For vector-valued coefficients: do the dimensions of sources and targets match? If one of these checks fails an exception is thrown and nothing happens (strong exception guarantee).

For maximum performance, you can omit these checks. If you pass data that violates the above criteria, however, the results will be undefined.

```
T.m2l_unchecked(translations.data(),translations.data()+translations.size());
```

Lastly, we would like to point out that it is possible to specify how many threads and which cores the `multithreaded_translator` should use. For this, you need to use `hwloc` and the following constructor:

```
multithreaded_translator( size_t P, hwloc_topology_t topo,
                          hwloc_cpuset_t *begin, hwloc_cpuset_t *end );
```

For each CPU-set that you pass, the `multithreaded_translator` creates and manages a thread and pins it to that set. How to create such sets is described in the `hwloc` documentation.

1.3.5 Computing Minimal Bounding Spheres

Since version 1.4, `solidfmm` provides routines for computing minimal bounding spheres. These are useful in adaptive fast multipole codes to avoid being overly pessimistic in the multipole acceptance criterion. We provide two kinds of routines: one computes the minimal bounding sphere of a set of points; the other routine can be used to compute the minimal bounding sphere of a set of spheres. To use these routines, include the following short, self-explanatory header:

```
#include <solidfmm/sphere.hpp>
```

The routines in this header implement an algorithm by Fischer and Gärtner.[8] The theoretical complexity of this algorithm is rather high. However, just like Fischer and Gärtner, in our experiments we found that the *empirical complexity is linear in the number points or spheres*. This implementation is particularly tuned for three-dimensional space; in our experiments it was faster than the more general CGAL implementation[2] and also provided more accurate results in floating point arithmetic.

[2] https://doc.cgal.org/latest/Bounding_volumes/classCGAL_1_1Min__sphere__d.html

Chapter 2

Mathematical Background

In this chapter we will give a description of some of the mathematical background of the fast multipole method (FMM) and its use of the solid harmonics. If you are already familiar with these topics, this chapter will probably contain nothing new for you, except for maybe sign and notational conventions.

2.1 Motivation: Movement of Planets

Consider we have a set of N planets in space, having location $\mathbf{x}_i = (x_i, y_i, z_i)^\top \in \mathbb{R}^3$ (measured in metres) and mass $m_i \in \mathbb{R}$ (measured in kilogrammes), $i = 1, \dots, N-1$. Throughout this text, we will use a **bold** font to indicate three-dimensional vectors. These planets mutually attract each other through gravity. Thus, let us define the gravimetric potential, following Newton's law of gravity:

$$\varphi(x,y,z) := G \sum_{i=0}^{N-1} \frac{m_i}{\sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2}} = G \sum_{i=0}^{N-1} \frac{m_i}{|\mathbf{x} - \mathbf{x}_i|}, \tag{2.1}$$

where $|\mathbf{x} - \mathbf{x}_i|$ denotes the Euclidean distance between the point $\mathbf{x} = (x,y,z)^\top \in \mathbb{R}^3$ and the location $\mathbf{x}_i$ of planet i, and $G \approx 6.674 \times 10^{-11} \frac{\mathrm{m}^3}{\mathrm{kg\,s}^2}$ is the universal gravitational constant. Then, applying Newton's laws of motion, the planet movements obey the following set of ordinary differential equations (ODEs):

$$\frac{\mathrm{d}^2 \mathbf{x}_i}{\mathrm{d}t^2} = -\nabla\varphi(\mathbf{x}_i) = \sum_{\substack{j=0 \\ j\neq i}}^{N-1} G m_j \frac{\mathbf{x}_j - \mathbf{x}_i}{|\mathbf{x}_j - \mathbf{x}_i|^3}, \qquad i = 0, \dots, N-1. \tag{2.2}$$

If you now consider entire galaxies, the number of planets is large, say $N \approx 1\,000\,000$. In order to solve this set of ODEs, you need to repeatedly evaluate $\nabla\varphi(\mathbf{x})$ at all planet locations $\mathbf{x}_i$. Each evaluation costs $\mathcal{O}(N)$, giving you a total cost of $\mathcal{O}(N^2) = \mathcal{O}(10^{12})$. This cost is prohibitive, even for large super computers. The fast multipole method (FMM) allows you to evaluate *approximations* of $\varphi(\mathbf{x})$ and its gradient and reduces this cost from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. This results in dramatic speed ups, and it is this speed up alone that makes large scale simulations of galaxies possible.

2.2 Expansions

2.2.1 Expansion of the Kernel Function

Let us denote by $k(\mathbf{x}, \mathbf{y}) \coloneqq |\mathbf{x} - \mathbf{y}|^{-1}$ the so-called *kernel function*. Using this function, the gravimetric potential from equation (2.1) can more compactly be given as

$$\varphi(\mathbf{x}) = G \sum_{i=0}^{N-1} m_i k(\mathbf{x}, \mathbf{x}_i). \tag{2.3}$$

The key to fast multipole methods is that this kernel function can be *expanded* in terms of the *solid harmonics* as follows:

$$k(\mathbf{x}, \mathbf{y}) \approx \sum_{n=0}^{P-1} \sum_{m=-n}^{n} S_n^m(\mathbf{x}) \overline{R_n^m(\mathbf{y})} \qquad \text{whenever } |\mathbf{x}| > |\mathbf{y}|. \tag{2.4}$$

The number $P \in \mathbb{N}$ is called the *order* of the expansion. The functions $S_n^m : \mathbb{R}^3 \setminus \{\mathbf{0}\} \to \mathbb{C}$ and $R_n^m : \mathbb{R}^3 \to \mathbb{C}$ are respectively called the *singular* and *regular solid harmonics*. While these functions are complex-valued—the overline on R_n^m denotes complex conjugation—the expansion always takes real values. We will define these functions precisely in following section. For now it suffices to remember that this expansion is very rapidly converging, it satisfies the following error-bound:[9, Theorem 3.2]

$$\left| k(\mathbf{x}, \mathbf{y}) - \sum_{n=0}^{P-1} \sum_{m=-n}^{n} S_n^m(\mathbf{x}) \overline{R_n^m(\mathbf{y})} \right| \leq \frac{1}{|\mathbf{x}| - |\mathbf{y}|} \left(\frac{|\mathbf{y}|}{|\mathbf{x}|} \right)^P. \tag{2.5}$$

In other words, its accuracy exponentially increases with order P, additionally the approximation gets more accurate as $|\mathbf{x}|$ increases and $|\mathbf{y}|$ decreases.

2.2.2 Multipole Expansions (P2M)

We now again consider the gravimetric potential φ as given in equation (2.3). Furthermore assume the case where planets are clustered around some point $\mathbf{x}_A \in \mathbb{R}^3$, and we would like to evaluate φ at locations $\mathbf{x}$ that are far away from that cluster, i. e., $|\mathbf{x} - \mathbf{x}_A| > |\mathbf{x}_i - \mathbf{x}_A|$ for all $i = 0, \dots, N-1$. We then have, by means of the kernel expansion (2.4):

$$\begin{aligned} \varphi(\mathbf{x}) &= \sum_{i=0}^{N-1} G m_i k(\mathbf{x}, \mathbf{x}_i) = \sum_{i=0}^{N-1} G m_i k(\mathbf{x} - \mathbf{x}_A, \mathbf{x}_i - \mathbf{x}_A) \\ &\overset{(2.4)}{\approx} \sum_{n=0}^{P-1} \sum_{m=-n}^{n} S_n^m(\mathbf{x} - \mathbf{x}_A) \overline{\underbrace{\sum_{i=0}^{N-1} R_n^m(\mathbf{x}_i - \mathbf{x}_A) G m_i}_{=:M_n^m}}. \end{aligned} \tag{2.6}$$

This suggests the following approach:

1. Compute the *multipoles* $M_n^m \in \mathbb{C}$:
$$M_n^m := \sum_{i=1}^{N} Gm_i R_n^m(\mathbf{x}_i - \mathbf{x}_A). \tag{2.7}$$
Cost: $\mathcal{O}(NP^2)$.

2. Evaluate the *multipole expansion* (or, if requested, its gradient) at the desired locations:
$$\varphi(\mathbf{x}) \approx \sum_{n=0}^{P-1} \sum_{m=-n}^{n} \overline{M_n^m} S_n^m(\mathbf{x}-\mathbf{x}_A) \qquad |\mathbf{x}-\mathbf{x}_A| > |\mathbf{x}_i-\mathbf{x}_A|, i = 1\ldots,N-1. \tag{2.8}$$
Cost: $\mathcal{O}\left(P^2\right)$ per evaluation point $\mathbf{x}$.

If we only want to evaluate φ at a single location $\mathbf{x}$, then nothing is gained. However, suppose you want to evaluate φ at M locations, where both $N, M \approx 1\,000\,000$. Then the cost for direct evaluation is $\mathcal{O}(NM) = \mathcal{O}\left(10^{12}\right)$, whereas the approach via the multipole expansion only costs $\mathcal{O}\left(NP^2 + MP^2\right) = \mathcal{O}\left(10^6 P^2\right)$. This results in the aforementioned dramatic speed up, assuming that P is not too large. In practice, choices up to $P = 20$ are common—remember that the kernel expansion (2.4) converges exponentially with P. Only very few calculations require even higher accuracies. Operation (2.7) computes multipole coefficients M_n^m from a given set of planets or particles; it is commonly called P2M: particle-to-multipole.

In summary, a multipole expansion approximates the potential φ of a particle cluster around some centre $\mathbf{x}_A$ *outside of this cluster*. The expansion gets *more accurate as the distance increases.* This is illustrated in Figure 2.1.

2.2.3 Local Expansions (P2L)

Local expansions in some ways consider the opposite case, as illustrated in Figure 2.2. Here we want to evaluate φ at a cluster of locations $\mathbf{x}$ around some centre $\mathbf{x}_B$, where the particles lie *outside* of this cluster. In other words, we have $|\mathbf{x} - \mathbf{x}_B| < |\mathbf{x}_i - \mathbf{x}_B|$ for all $i = 1, \ldots, N-1$, and thus:

$$\begin{aligned}\varphi(\mathbf{x}) &= \sum_{i=0}^{N-1} Gm_i k(\mathbf{x}, \mathbf{x}_i) = \sum_{i=0}^{N-1} Gm_i k(\mathbf{x}_i - \mathbf{x}_B, \mathbf{x} - \mathbf{x}_B) \\ &\overset{(2.4)}{\approx} \sum_{n=0}^{P-1} \sum_{m=-n}^{n} \overline{R_n^m(\mathbf{x} - \mathbf{x}_B)} \underbrace{\sum_{i=0}^{N-1} S_n^m(\mathbf{x}_i - \mathbf{x}_B) Gm_i}_{=:L_n^m}.\end{aligned} \tag{2.9}$$

After computing the *local coefficients* $L_n^m \in \mathbb{C}$, the potential φ can be approximated near $\mathbf{x}_B$ using the local expansion:

$$\varphi(\mathbf{x}) \approx \sum_{n=0}^{P-1} \sum_{m=-n}^{n} L_n^m \overline{R_n^m(\mathbf{x} - \mathbf{x}_B)} \qquad |\mathbf{x} - \mathbf{x}_B| < |\mathbf{x}_i - \mathbf{x}_B|,\ i = 1, \ldots, N-1. \tag{2.10}$$

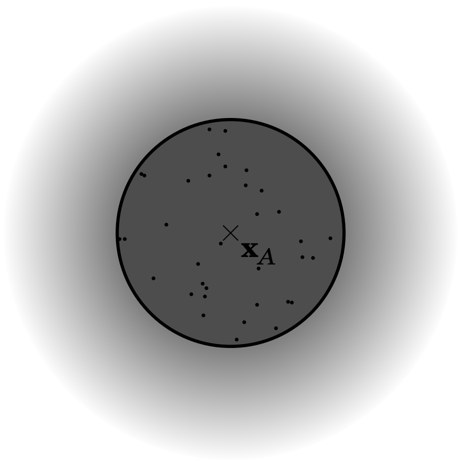


Figure 2.1: Multipole expansions converge outside of the smallest ball around their centre that contains all particles. Here, the cluster of planets near the centre $\mathbf{x}_A$ is contained within the drawn circle. Outside this region the expansion converges and gets more accurate as the distance increases, indicated by grey fading into white. The accuracy increases faster for expansions of higher order P. The expansion does not converge within the circle and gives large errors there, indicated by a dark grey colour.

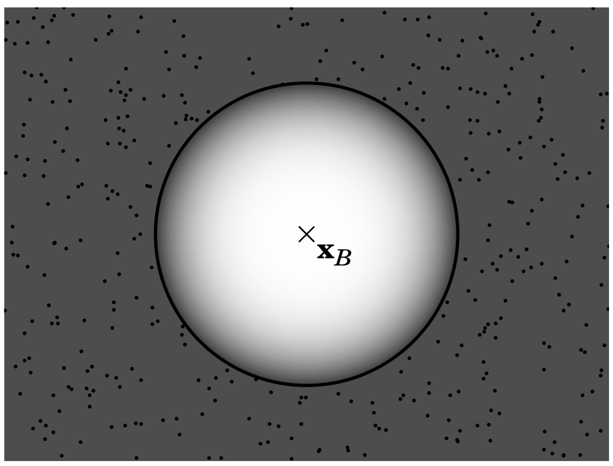


Figure 2.2: Local expansions converge in the largest ball around their centre that contains no particles. The error of a local expansion is zero at its centre $\mathbf{x}_B$, indicated by a shade of white in this figure. It then gradually increases when approaching the boundary of the ball (white fading to grey). This increase is slower for expansions of higher order P. Outside of this ball, the local expansion does not converge and gives large errors (dark grey).

The operation of computing the local coefficients directly from the particles is called particle-to-local (P2L):

$$L_n^m := \sum_{i=0}^{N-1} S_n^m(\mathbf{x}_i - \mathbf{x}_B) G m_i. \tag{2.11}$$

However, most FMM implementations do not compute local coefficients via P2L; those that do only use it in rare circumstances. The usual way to obtain local coefficients is the so-called M2L translation and will be discussed later.

2.2.4 Remarks on the Derivation of Expansions

Key to all fast multipole methods is the availability of rapidly converging series expansions of the underlying kernel function $k(\mathbf{x}, \mathbf{y})$, such as in Equation (2.4). Generic expansions, such as Taylor series, can in principle be applied to any kind of kernel k. For example, the Taylor expansion of order P of some kernel $k(\mathbf{x}, \mathbf{y})$ for a fixed $\mathbf{x}$ around $\mathbf{y} = \mathbf{0}$ reads:

$$k(\mathbf{x}, \mathbf{y}) \approx \sum_{|\boldsymbol{\alpha}|<P} \underbrace{\partial_{\mathbf{y}}^{\boldsymbol{\alpha}} k(\mathbf{x}, \mathbf{0})}_{=:\widetilde{S}_{\boldsymbol{\alpha}}(\mathbf{x})} \underbrace{\frac{\mathbf{y}^{\boldsymbol{\alpha}}}{\boldsymbol{\alpha}!}}_{=:\widetilde{R}_{\boldsymbol{\alpha}}(\mathbf{y})}, \tag{2.12}$$

where $\boldsymbol{\alpha} \in \mathbb{N}_0^3$ is a multi-index. The key point is that this expansion is a sum of products, where one factor only depends on $\mathbf{x}$ and the other factor only depends on $\mathbf{y}$. In this way the simultaneous dependence of k on both $\mathbf{x}$ and $\mathbf{y}$ *decouples*.

The Taylor expansion consists of $\mathcal{O}(P^3)$ terms, where the functions $\widetilde{S}_{\boldsymbol{\alpha}}$ and $\widetilde{R}_{\boldsymbol{\alpha}}$ take similar roles as the S_n^m and R_n^m in Equation (2.4). Thus, assuming sufficient regularity of k, Taylor's theorem is all we need to to construct multipole and local expansions and it also provides us with error bounds. Without additional knowledge about k, this is essentially already the best we can do. However, also note that the Taylor expansion has $\mathcal{O}(P^3)$ terms, while Equation (2.4) requires only $\mathcal{O}(P^2)$. Why is this so?

With the particular choice $k(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}|^{-1}$ and fixed $\mathbf{x}$, the Taylor polynomials (2.12) are always *harmonic*. In other words, the exact kernel function satisfies $-\Delta_{\mathbf{y}} k(\mathbf{x}, \mathbf{y}) = 0$, where $-\Delta_{\mathbf{y}}$ is the Laplace operator with respect to the $\mathbf{y}$-variable. This property is inherited by the Taylor polynomials. There are only $\mathcal{O}(P^2)$ linearly independent harmonic polynomials, so it suffices to use a basis for this *sub*space. The regular solid harmonics R_n^m, $|m| \leq n$, are exactly that: they are a basis of the space of homogeneous harmonic polynomials of degree n. Expansion (2.4) achieves better compression because only basis functions for the proper subspace are used. The Taylor expansion (2.12), on the other hand, uses the generic monomial basis for *all* polynomials, not just the harmonic ones.

Only for the particular kernel $k(\mathbf{x}, \mathbf{y}) = |\mathbf{x} - \mathbf{y}|^{-1}$ the solid harmonics are the correct choice. `solidfmm` only considers this kernel function and the solid harmonics. For other kernel functions one can either use the more expensive, but more general Taylor expansion (2.12), or other generic $\mathcal{O}(P^3)$ approaches like Chebyshev interpolation.

Alternatively, if performance is critical, it might be worthwhile to develop more efficient expansions specific to the particular kernel at hand.

2.3 Solid Harmonics

The solid harmonics can be motivated and derived in many different ways. We refer the interested reader to the literature for this. Often they are given in spherical coordinates. When restricting the solid harmonics to the surface of the unit sphere, one obtains the more well-known *spherical* harmonics $Y_n^m(\theta,\varphi)$, i. e., $Y_n^m(\theta,\varphi) = R_n^m(r=1,\theta,\varphi)$. This explains their name: unlike the spherical harmonics Y_n^m, the solid harmonics S_n^m and R_n^m are defined not only on the sphere's surface, but also on the volume it encloses, i. e., on the entire solid sphere. They are called *harmonic* because $-\Delta S_n^m(\mathbf{x}) = 0$ for $\mathbf{x} \in \mathbb{R}^3 \setminus \{\mathbf{0}\}$ and $-\Delta R_n^m(\mathbf{x}) = 0$ for $\mathbf{x} \in \mathbb{R}^3$. The functions S_n^m are called singular solid harmonics because of their singularities at $\mathbf{x} = \mathbf{0}$. The functions R_n^m are polynomials, hence they are called regular solid harmonics.

While spherical coordinates have certain benefits for analysis, these functions can more easily evaluated in Cartesian coordinates. There are countless conventions concerning signs and normalisations. `solidfmm` follows the convention given by Dehnen.[10] For computational purposes, the solid harmonics are best defined recursively. Let $\mathbf{x} = (x,y,z)^\top \in \mathbb{R}^3$ be given, and let $|\mathbf{x}| = \sqrt{x^2+y^2+z^2}$ denote its Euclidean length. Starting with $S_0^0(\mathbf{x}) = |\mathbf{x}|^{-1}$ and $R_0^0(\mathbf{x}) = 1$, one continues with the diagonal $n = m$:

$$S_n^n = (2n-1)\frac{x+Iy}{|\mathbf{x}|^2}S_{n-1}^{n-1}, \qquad R_n^n = \frac{x+Iy}{2n}R_{n-1}^{n-1}, \tag{2.13}$$

where I denotes the imaginary unit $I^2 = -1$. For the remaining non-negative $m = 0,\dots,n-1$, they can be computed via:

$$\begin{aligned} |\mathbf{x}|^2 S_n^m &= (2n-1)zS_{n-1}^m - ((n-1)^2-m^2)S_{n-2}^m, \\ (n^2-m^2)R_n^m &= (2n-1)zR_{n-1}^m - |\mathbf{x}|^2 R_{n-2}^m. \end{aligned} \tag{2.14}$$

Here, it is implicitly assumed that $R_{n-2}^m = S_{n-2}^m = 0$ whenever $m = n-1$.

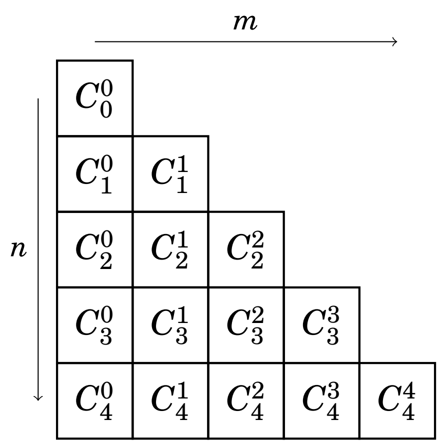
For the negative values of m, i. e., $m = -n,\dots,-1$, they are defined *implicitly* via

$$S_n^m(\mathbf{x}) = (-1)^m\overline{S_n^{-m}(\mathbf{x})} \quad \text{and} \quad R_n^m(\mathbf{x}) = (-1)^m\overline{R_n^{-m}(\mathbf{x})}. \tag{2.15}$$

This property therefore also holds for the coefficients of the multipole and local expansions:

$$M_n^m = (-1)^m\overline{M_n^{-m}} \quad \text{and} \quad L_n^m = (-1)^m\overline{L_n^{-m}}. \tag{2.16}$$

Thus, to even further increase compression, in a computer implementation neither the solid harmonics R_n^m and S_n^m, nor the multipole and local coefficients L_n^m and M_n^m should be explicitly stored for negative m. Taking these symmetries into account,



m
n
C_0^0
C_1^0 C_1^1
C_2^0 C_2^1 C_2^2
C_3^0 C_3^1 C_3^2 C_3^3
C_4^0 C_4^1 C_4^2 C_4^3 C_4^4

Figure 2.3: The coefficients $C_n^m \in \mathbb{C}$, $C \in \{S, R, M, L\}$ of both multipole and local expansions, as well as the values of the solid harmonics S_n^m and R_n^m can be stored using $P(P+1)/2$ complex numbers. In this example we have $P = 5$. They can be arranged in a triangular pattern according to their indices $n = 0, \ldots, 4$ and $m = 0, \ldots, n$, where the values for negative m are given implicitly by $C_n^m = (-1)^m \overline{C_n^{-m}}$.

a local or multipole expansion of order P consists of $P(P+1)/2$ complex numbers, which can be arranged in a triangular pattern as shown in Figure 2.3.

At the same time, this property guarantees that the expansion (2.4) always is real-valued: for $m = 0$ we have $R_n^0 \in \mathbb{R}$ and $S_n^0 \in \mathbb{R}$. For $m \neq 0$ the imaginary parts for $\pm m$ cancel each other out:

$$\sum_{n=0}^{P-1} \sum_{m=-n}^{n} S_n^m \overline{R_n^m} = \sum_{n=0}^{P-1} S_n^0 R_n^0 + \sum_{n=0}^{P-1} \sum_{m=1}^{n} \left(S_n^m \overline{R_n^m} + S_n^{-m} \overline{R_n^{-m}} \right)$$
$$\overset{(2.15)}{=} \sum_{n=0}^{P-1} \underbrace{S_n^0 R_n^0}_{\in \mathbb{R}} + \sum_{n=0}^{P-1} \sum_{m=1}^{n} \underbrace{\left(S_n^m \overline{R_n^m} + \overline{S_n^m} R_n^m \right)}_{\in \mathbb{R}}. \qquad (2.17)$$

2.4 Translations

Translation operators take in one expansion around centre $\mathbf{x}_A$ and produce another around some centre $\mathbf{x}_B$. When both in- and output expansions are of the same order P, the translation operators map $\mathcal{O}(P^2)$ input coefficients to $\mathcal{O}(P^2)$ output coefficients at the cost of $\mathcal{O}(P^4)$ arithmetic operations. The so-called M2L-translation is particularly important for FMMs, and a significant portion of the computational time is spent on performing these translations. They are the main reason for the existence of this library: it provides highly efficient, vectorised implementations of the translation operators. Additionally, `solidfmm`'s implementation is based on the approach outlined by Dehnen,[10] which reduces the complexity from $\mathcal{O}(P^4)$ to $\mathcal{O}(P^3)$ in theory and even $\mathcal{O}(P^2)$ in practice.

2.4.1 Multipole-to-Multipole (M2M)

M2M may be interpreted just like P2M, with the difference that the input is not a planet with mass m_i at location $\mathbf{x}_i$, but a multipole expansion of order P_i with coefficients $M_{n,i}^m$ and centre $\mathbf{x}_i$. Thus, in M2M, both the input and output are multipole expansions. We again denote by $\mathbf{x}_A$ and P the centre and order of the output expansion. We furthermore introduce the *shift vector* $\mathbf{r} = \mathbf{x}_A - \mathbf{x}_i$, pointing from the old to the new expansion centre.

Instead of expanding the kernel $k(\mathbf{x}, \mathbf{y}) = S_0^0(\mathbf{x} - \mathbf{y})$, we now expand the singular solid harmonics $S_n^m(\mathbf{x} - \mathbf{y})$, analogously to expansion (2.4):[10, Equation (49)]

$$S_n^m(\mathbf{x} - \mathbf{y}) = \sum_{k=0}^{\infty} \sum_{l=-k}^{k} S_{n+k}^{m+l}(\mathbf{x}) \overline{R_k^l(\mathbf{y})} \qquad |\mathbf{x}| > |\mathbf{y}|. \tag{2.18}$$

Thus, whenever $|\mathbf{x} - \mathbf{x}_A| > |\mathbf{x}_i - \mathbf{x}_A|$, the input multipole expansion with centre $\mathbf{x}_i$ can itself be expanded around $\mathbf{x}_A$ as follows:

$$\begin{aligned}
&\sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} \overline{M_{n,i}^m} S_n^m(\mathbf{x} - \mathbf{x}_i) \\
&\qquad = \sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} \overline{M_{n,i}^m} S_n^m((\mathbf{x} - \mathbf{x}_A) - (\mathbf{x}_i - \mathbf{x}_A)) \\
&\qquad \overset{(2.18)}{=} \sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} \sum_{k=0}^{\infty} \sum_{l=-k}^{k} \overline{M_{n,i}^m R_k^l(-\mathbf{r})} S_{n+k}^{m+l}(\mathbf{x} - \mathbf{x}_A).
\end{aligned} \tag{2.19}$$

All that remains is a change of indices and rearranging the sums. To simplify the summation limits, we will implicitly assume that we have $R_{n-k}^{m-l}(-\mathbf{r}) = 0$ whenever $|m - l| > n - k$ or $n - k < 0$. This finally results in:

$$\begin{aligned}
&\sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} \overline{M_{n,i}^m} S_n^m(\mathbf{x} - \mathbf{x}_i) \\
&\qquad = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} S_n^m(\mathbf{x} - \mathbf{x}_A) \left(\sum_{k=0}^{P_i-1} \sum_{l=-k}^{k} \overline{M_{k,i}^l R_{n-k}^{m-l}(-\mathbf{r})} \right).
\end{aligned} \tag{2.20}$$

We may thus define the new multipoles M_n^m via:

$$M_n^m := \sum_{k=0}^{P_i-1} \sum_{l=-n}^{n} M_{k,i}^l R_{n-k}^{m-l}(-\mathbf{r}). \tag{2.21}$$

This is the general M2M-translation formula. We then have, after truncating at the given output order P:

$$\sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} \overline{M_{n,i}^m} S_n^m(\mathbf{x} - \mathbf{x}_i) \approx \sum_{n=0}^{P-1} \sum_{m=-n}^{n} \overline{M_n^m} S_n^m(\mathbf{x} - \mathbf{x}_A) \qquad |\mathbf{x} - \mathbf{x}_A| > |\mathbf{x}_i - \mathbf{x}_A|. \tag{2.22}$$

While P2M allows us to combine several 'particles' or 'planets' into a single multipole expansion with centre $\mathbf{x}_A$, M2M allows us to combine several multipole expansions into one. Suppose we were given N multipole expansions with centres $\mathbf{x}_i$ and orders P_i, $i = 0, \dots, N-1$. The potential approximated by these expansions:

$$\varphi(\mathbf{x}) \approx \sum_{i=0}^{N-1} \sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} \overline{M_{n,i}^m} S_n^m(\mathbf{x} - \mathbf{x}_i) \tag{2.23}$$

can then in turn be approximated as follows:

1. Compute the combined multipole moments around $\mathbf{x}_A$ using M2M:

$$M_n^m \overset{(2.21)}{=} \sum_{i=0}^{N-1} \sum_{k=0}^{P_i-1} \sum_{l=-k}^{k} M_{k,i}^l R_{n-k}^{m-l}(\mathbf{x}_i - \mathbf{x}_A). \tag{2.24}$$

   Cost, assuming $P_i = P$ for all $i = 0, \dots, N-1$: $\mathcal{O}\left(NP^4\right)$. `solidfmm` reduces this cost to $\mathcal{O}\left(NP^3\right)$ for large P. For all practically relevant choices of P, however, the empirically measured timings behave even better, namely optimally as $\mathcal{O}\left(NP^2\right)$.

2. Evaluate the new expansion at the desired output locations:

$$\varphi(\mathbf{x}) \approx \sum_{n=0}^{P-1} \sum_{m=-n}^{n} \overline{M_n^m} S_n^m(\mathbf{x} - \mathbf{x}_A). \tag{2.25}$$

   Cost: $\mathcal{O}\left(P^2\right)$ per evaluation point $\mathbf{x}$.

The resulting approximation now contains *two sources of error*:

1. The error of the original approximation (2.23).
2. The error from the M2M translation.

The additional error behaves just like the error for an individual expansion obtained via P2M. In particular, Figure 2.1 also describes this additional error and area of convergence, where the 'dots' in this figure now correspond to the multipole expansions $M_{n,i}^m$ and their centres $\mathbf{x}_i$.

2.4.2 Multipole-to-Local (M2L)

M2L is to P2L what M2M is to P2M: we can again treat a given multipole expansion just like a planet or particle. While P2L allows us to locally approximate the potential induced by several planets, M2L allows us to approximate the potential given by multipole expansions. Thus, suppose we were given a multipole expansion with coefficients $M_{n,i}^m$ and order P_i around some centre $\mathbf{x}_i$. We now wish to locally approximate this expansion near $\mathbf{x}_B$ using a local expansion. Again, we denote by $\mathbf{r} = \mathbf{x}_B - \mathbf{x}_i$ the *shift vector*, pointing from the old to the new expansion centre.

We will make use of the fact that we can flip signs via $S_n^m(-\mathbf{x}) = (-1)^n S_n^m(\mathbf{x})$. For $|\mathbf{x} - \mathbf{x}_B| < |\mathbf{x} - \mathbf{x}_i|$ we then have:

$$\begin{aligned}
&\sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} \overline{M_{n,i}^m} S_n^m(\mathbf{x} - \mathbf{x}_i) \\
&\quad= \sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} \overline{M_{n,i}^m} S_n^m\left((\mathbf{x} - \mathbf{x}_B) - (\mathbf{x}_i - \mathbf{x}_B)\right) \\
&\quad\overset{\text{sign flip}}{=} \sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} (-1)^n \overline{M_{n,i}^m} S_n^m\left((\mathbf{x}_i - \mathbf{x}_B) - (\mathbf{x} - \mathbf{x}_B)\right) \\
&\quad\overset{(2.18)}{=} \sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} \sum_{k=0}^{\infty} \sum_{l=-k}^{k} (-1)^n \overline{M_{n,i}^m} S_{n+k}^{m+l}(-\mathbf{r}) \overline{R_k^l(\mathbf{x} - \mathbf{x}_B)} \\
&\quad\overset{\text{sign flip}}{=} \sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} \sum_{k=0}^{\infty} \sum_{l=-k}^{k} (-1)^k \overline{M_{n,i}^m} S_{n+k}^{m+l}(\mathbf{r}) \overline{R_k^l(\mathbf{x} - \mathbf{x}_B)}. \qquad (2.26)
\end{aligned}$$

It again remains to rearrange the sums and change the indices to obtain:

$$\begin{aligned}
&\sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} \overline{M_{n,i}^m} S_n^m(\mathbf{x} - \mathbf{x}_i) \\
&\quad= \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \overline{R_n^m(\mathbf{x} - \mathbf{x}_B)} \left((-1)^n \sum_{k=0}^{P_i-1} \sum_{l=-k}^{k} \overline{M_{k,i}^l} S_{n+k}^{m+l}(\mathbf{r}) \right). \qquad (2.27)
\end{aligned}$$

Thus, the general M2L-translation formula is as follows:

$$L_n^m := (-1)^n \sum_{k=0}^{P_i-1} \sum_{l=-k}^{k} \overline{M_{k,i}^l} S_{n+k}^{m+l}(\mathbf{r}). \qquad (2.28)$$

After truncating at the output order P, the resulting local expansion then approximates the original multipole expansion near $\mathbf{x}_B$:

$$\sum_{n=0}^{P_i-1} \sum_{m=-n}^{n} \overline{M_n^m} S_n^m(\mathbf{x} - \mathbf{x}_i) \approx \sum_{n=0}^{P-1} \sum_{m=-n}^{n} L_n^m \overline{R_n^m(\mathbf{x} - \mathbf{x}_B)} \qquad |\mathbf{x} - \mathbf{x}_B| < |\mathbf{x}_i - \mathbf{x}_B|. \qquad (2.29)$$

The remarks on the error of M2M analogously carry over to M2L. Several multipole expansions can be combined into a single local expansion. There are then two sources of error: the error from the original expansions plus the error from M2L. The second error contribution behaves just like in Figure 2.2: it is zero at $\mathbf{x}_B$, and then gradually increases as one approaches the perimeter of the largest sphere around $\mathbf{x}_B$ that contains none of the $\mathbf{x}_i$.

2.4.3 Local-to-Local (L2L)

L2L differs from the previous translation operators in that it does not make sense to choose the output's order P higher than that of the input expansion P_i. In L2L, the input is a local expansion of order P_i, i. e., a harmonic polynomial of total degree less than P_i. The space of harmonic polynomials is invariant under translation, i. e., polynomials can be translated exactly. Thus, when $P = P_i$, L2L is an error-free operation. Errors are only introduced when $P < P_i$, in which case they are zero at the new expansion centre $\mathbf{x}_B$ and gradually increase from there.

The derivation is similar to that of M2M and M2L. We begin by expanding the regular solid harmonics:[10, Equation (48)]

$$R_n^m(\mathbf{x}-\mathbf{y}) = \sum_{k=0}^{n}\sum_{l=-k}^{k} R_k^l(\mathbf{x})R_{n-k}^{m-l}(-\mathbf{y}) \qquad \text{(exact for any } \mathbf{x},\mathbf{y}\in\mathbb{R}^3\text{)}. \tag{2.30}$$

Here, to simplify notation, we again implicitly assume that $R_n^m = 0$ whenever $|m| > n$.

Thus, for a given input expansion $L_{n,i}^m$ around $\mathbf{x}_i$ and of order P_i, and shift vector $\mathbf{r} = \mathbf{x}_B - \mathbf{x}_i$, one obtains:

$$\begin{aligned}\sum_{n=0}^{P_i-1}\sum_{m=-n}^{n} L_{n,i}^m\overline{R_n^m(\mathbf{x}-\mathbf{x}_i)} &= \sum_{n=0}^{P_i-1}\sum_{m=-n}^{n} L_{n,i}^m\overline{R_n^m((\mathbf{x}-\mathbf{x}_B)-(\mathbf{x}_i-\mathbf{x}_B))}\\ &\overset{(2.30)}{=} \sum_{n=0}^{P_i-1}\sum_{m=-n}^{n}\sum_{k=0}^{n}\sum_{l=-k}^{k} L_{n,i}^m\overline{R_k^l(\mathbf{x}-\mathbf{x}_B)R_{n-k}^{m-l}(\mathbf{r})}\\ &\overset{\text{rearranging}}{=} \sum_{n=0}^{P_i-1}\sum_{m=-n}^{n}\overline{R_n^m(\mathbf{x}-\mathbf{x}_B)}\left(\sum_{k=n}^{P_i-1}\sum_{l=-k}^{k} L_{k,i}^l\overline{R_{k-n}^{l-m}(\mathbf{r})}\right).\end{aligned} \tag{2.31}$$

Thus, we may define new local moments via:

$$L_n^m = \sum_{k=n}^{P_i-1}\sum_{l=-k}^{k} L_{k,i}^l\overline{R_{k-n}^{l-m}(\mathbf{r})} \tag{2.32}$$

resulting in

$$\sum_{n=0}^{P_i-1}\sum_{m=-n}^{n} L_{n,i}^m\overline{R_n^m(\mathbf{x}-\mathbf{x}_i)} = \sum_{n=0}^{P_i-1}\sum_{m=-n}^{n} L_n^m\overline{R_n^m(\mathbf{x}-\mathbf{x}_B)} \qquad \text{exactly.} \tag{2.33}$$

The output expansion may be truncated ($P < P_i$). Choosing ($P > P_i$) will only result in adding zero coefficients $L_n^m = 0$ for $n \geq P_i$.

2.4.4 Summary of the Translation Formulæ

All of the translation operators take an input expansion of order P_i around some centre $\mathbf{x}_i$ and produce a new expansion of order P around centre $\mathbf{x}_i + \mathbf{r}$, where $\mathbf{r}$ is the so-called *shift vector*. Depending on the type of translation, the output expansion may coincide with the input expansion, or just be an approximation of it. The formulæ for computing the output coefficients are as follows.

M2M:

$$M_n^m = \sum_{k=0}^{P_i-1} \sum_{l=-n}^{n} M_{k,i}^l R_{n-k}^{m-l}(-\mathbf{r}). \tag{2.34}$$

M2L:

$$L_n^m = (-1)^n \sum_{k=0}^{P_i-1} \sum_{l=-k}^{k} \overline{M_{k,i}^l} S_{n+k}^{m+l}(\mathbf{r}). \tag{2.35}$$

L2L:

$$L_n^m = \sum_{k=n}^{P_i-1} \sum_{l=-k}^{k} L_{k,i}^l \overline{R_{k-n}^{l-m}(\mathbf{r})}. \tag{2.36}$$

In these sums it is implicitly assumed that $R_n^m = 0$ whenever $n < 0$ or $|m| > n$. When in- and output order $P = P_i$ coincide, computing all output coefficients using these forumulæ costs $\mathcal{O}(P^4)$. `solidfmm` uses acceleration techniques to reduce this cost to $\mathcal{O}(P^3)$ in theory, but measurements have shown that in practice it performs even better as $\mathcal{O}(P^2)$.

2.5 Outlook

The concepts described in this chapter form the main building blocks of fast multipole methods. With the mathematical background presented here, it should be possible to understand the library functionality described in Chapter 1. At the moment, `solidfmm` does not provide a full implementation of the fast multipole method, but only an efficient, optimised implementation of these building blocks. This is intentional, so that the library remains slim can be used in existing implementations of the FMM. For this reason we do not give a full description of the algorithm, of which there exist many variations.

The original FMM is due to Greengard and Rokhlin.[11] Since then the algorithm has developed into more simple, adaptive, parallelisable variants. A description of the 'dual tree traversal' by Dehnen[12] and a comparison to other variants was given by Yokota.[13] We find this version of the FMM particularly intuitive. He also compares different series expansions and mentions the lack of efficient implementations for the solid harmonics, even though they are the most efficient expansions available for the important kernel function $k(\mathbf{x},\mathbf{y}) = |\mathbf{x}-\mathbf{y}|^{-1}$. `solidfmm`'s aim is to provide such an implementation.

Chapter 3

User Reference

This chapter considers `solidfmm`'s API for users. The implementation details are covered in Part II of this book. All components of the library are in the `solidfmm` namespace. For brevity, we will omit the explicitly stating namespace in the discussion that follows. For example we will simply write `solid` instead of `solidfmm::solid`.

3.1 Overview

`solidfmm` exposes its interface through four header files, each of which will be described in a separate section below:

- `solidfmm/solid.hpp` Contains the `solid` data structure for storing the coefficients of expansions and values of the solid harmonics. Also contains the `dot` and `fmadd` operations.
- `solidfmm/harmonics.hpp` Evaluation of the solid harmonics and their gradients.
- `solidfmm/handles.hpp` Handle classes for creating buffers and shared data which are needed for the translation operations.
- `solidfmm/translations.hpp` The actual translation operations.

Most operations are templates and take a type parameter which is commonly called `real`. This type specifies which fundamental arithmetic type should be used to represent a real number. `solidfmm` supports single and double precision, i. e., `float` and `double` for this parameter. It is illegal to use other types for `real`.

3.2 `solidfmm/solid.hpp`

Objects of type `solid` are used to store the coefficients of both local and multipole expansions. Additionally, when evaluating the solid harmonics $S_n^m(\mathbf{x})$ and $R_n^m(\mathbf{x})$ at some given point $\mathbf{x} \in \mathbb{R}^3$, the resulting values are also stored in objects of this type. Instances can either hold scalar or vector-valued data. For example, the solid harmonics $S_n^m(\mathbf{x}), R_n^m(\mathbf{x}) \in \mathbb{C}$ are scalars. Their gradients are vector-valued: $\nabla S_n^m(\mathbf{x}), \nabla R_n^m(\mathbf{x}) \in \mathbb{C}^3$. For these reason most operations additionally take a parameter called `dim`, which, however, can be omitted when working with scalars. Similarly, it is possible to use vector-valued multipole and local expansions.

```
template <typename real>
class solid
{
public:
    solid() = default;
    solid( size_t P, size_t dim = 1 );
    solid( const solid  &rhs );
    solid(       solid &&rhs ) noexcept;
    solid& operator=( const solid  &rhs );
    solid& operator=(       solid &&rhs ) noexcept;
   ~solid();

    void resize( size_t P, size_t dim = 1 );
    void reinit( size_t P, size_t dim = 1 );
    void zeros() noexcept;

    const real& re( size_t n, size_t m, size_t d = 0 ) const noexcept;
          real& re( size_t n, size_t m, size_t d = 0 )       noexcept;
    const real& im( size_t n, size_t m, size_t d = 0 ) const noexcept;
          real& im( size_t n, size_t m, size_t d = 0 )       noexcept;

    size_t order    () const noexcept;
    size_t dimension() const noexcept;

    const real*  memptr() const noexcept;
          real*  memptr()       noexcept;

private:
        // …
};
```

Listing 3.1: Public interface of the `solid` class.

3.2.1 Construction and Resizing

The default constructor creates instances of order $P = 0$, i. e., empty objects which do not contain any coefficients or values. When creating a fresh object using the second constructor, or when calling `reinit` with the desired order and dimension, one obtains a `solidfmm::solid` containing only zero values. Calling the member `resize`, on the other hand, will result in object of the desired order and dimension, but with *undefined* contents. `zeros` can be used to set all members to zero. Thus, `reinit` is equivalent to calling `resize` and `zeros` immediately afterwards. The copy and move constructors, as well as the assignment operators have the usual semantics. All these operations either come with a no-throw guarantee, or the strong exception guarantee. In particular, functions like `resize` can only throw exceptions of type `std::bad_alloc`, but in case they do, they leave the object unchanged.

3.2.2 Accessing Elements

The contents of a solid can be accessed using the member functions re and im, which respectively return the real and imaginary parts of the specified element.

```
solid<double> M; // Given multipole coefficients, initialised somewhere else.

double a = M.re(4,2);
M.im(2,1) = 42;
```

Under the assumption that M.order() > 4, this code stores the value of $\Re M_4^2$ in the variable a and assigns the value 42 to $\Im M_2^1$. For vector-valued a solid, the above code would access the zeroth component of the respective vectors; dimensions are counted beginning with zero. For example, to access the 'y-component' of the real part of some $\mathbf{M}_4^2 \in \mathbb{C}^3$, one would instead use:

```
double a = M.re(4,2,1);
```

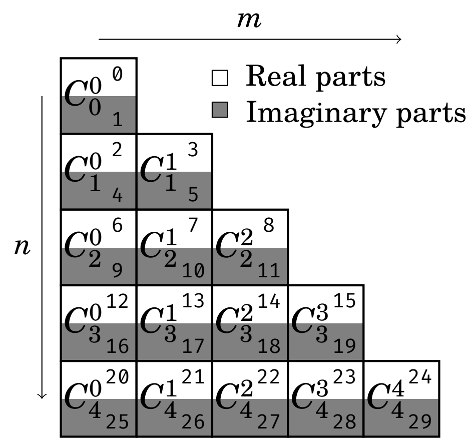
The indices n, m, and d (default value $d = 0$) must be valid. No range checks are performed; passing invalid arguments will result in undefined behaviour. As discussed in Section 2.3, it is not necessary to store the values of M_n^m for the negative values of m, as they are given implicitly. For this reason, for a given solid of order P and dimension D, the valid range of parameters is $0 \leq n < P$, $0 \leq m \leq n$, and $0 \leq d < \mathrm{D}$.

3.2.3 Direct Memory Access

The member function memptr() gives direct memory access to the data that a solid is holding. The indexing scheme is best illustrated graphically, see Figure 3.1. Assume we were given a solid of order P and dimension D. Then the real and imaginary parts of the dth vector-component of some $\mathbf{M}_n^m \in \mathbb{C}^{\mathrm{D}}$, $0 \leq n < P$, $0 \leq m \leq n$, $0 \leq d < \mathrm{D}$ can be accessed via:

```
double real_part = M.memptr()[ d*P*(P+1) + (n  )*(n+1) + m ];
double imag_part = M.memptr()[ d*P*(P+1) + (n+1)*(n+1) + m ];
```

Don't even think about calling delete, delete [], free(), realloc(), or the like on this pointer—unless, of course, you are keen on causing undefined behaviour and crashing your programme.



$$\begin{aligned}\texttt{pos}(\Re(C_n^m)) &= (n+0)(n+1)+m,\\ \texttt{pos}(\Im(C_n^m)) &= (n+1)(n+1)+m.\end{aligned}$$

Figure 3.1: A generic `solid` C may contain coefficients of local or multipole expansions, as well as the values of regular or singular harmonics. The memory layout of a *scalar* `solid` of order 5 is illustrated above. Only the values C_n^m for $m \geq 0$ need to be stored, as the values for negative m are given implicitly. The array position `pos` of the respective values is indicated by the numbers in the upper and lower right corners of the boxes. A vector-valued `solid` first stores all values for dimension $d = 0$ as shown above. Then all elements for dimension $d = 1$ follow using the same layout, then those of dimension $d = 2$, and so forth.

3.2.4 Evaluating Expansions using dot

Consider the task of evaluating a given multipole expansion around some centre $\mathbf{x}_A$:

$$\sum_{n=0}^{P-1} \sum_{m=-n}^{n} \overline{M_n^m} S_n^m(\mathbf{x} - \mathbf{x}_A) \tag{3.1}$$

as discussed in Chapter 2. This operation is essentially the dot-product on two complex-valued vectors $M, S \in \mathbb{C}^{P^2}$ which are indexed in a non-standard way. Additionally, as discussed before, this sum always takes real values, so the roles of M and S can be swapped without changing the result. For this reason, `solidfmm` implements such operations using the function `dot`.

```
void dot( const solid<float>  &A, const solid<float>  &B, float  *result ) noexcept;
void dot( const solid<double> &A, const solid<double> &B, double *result ) noexcept;
```

This function supports inputs of differing orders and dimensions, i. e., it is legal to have `A.order()` $\neq$ `B.order()` and `A.dimension()` $\neq$ `B.dimension()`. The semantics of these cases are as follows.

1. In case we have `A.order()` $\neq$ `B.order()`, the `solid` of lower order is padded with zeros to match the other `solid`'s order.

2. In case we have `A.dimension()`$\neq$`1` or `B.dimension()`$\neq$`1`, all combinations of components are dotted together, i. e., the result is a matrix of dimension

   $$\texttt{A.dimension()} \times \texttt{B.dimension()}$$

   that is stored in row-major order. The entry at position (i,j) then contains the value

   $$\sum_{n=0}^{P-1} \sum_{m=-n}^{n} \overline{A_{n,i}^m} B_{n,j}^m.$$

   If you prefer column-major order, you can just swap `A` and `B` when calling `dot`.

For this reason, `dot` does not simply return a floating point value, but takes a pointer to the first element where the result should be stored. It is the duty of the user to ensure that these pointers point to a region of memory which is sufficiently large to store the result.

Assume you were given a vector-valued local expansion of order P with coefficients $\mathbf{L}_n^m \in \mathbb{C}^3$ around expansion centre $\mathbf{x}_B = (\texttt{xB}, \texttt{yB}, \texttt{zB})^\top \in \mathbb{R}^3$. To evaluate this expansion at some location $\mathbf{x} = (\texttt{x}, \texttt{y}, \texttt{z})^\top \in \mathbb{R}^3$, you would use the following code:

```
double result[3];
dot( L, harmonics::R<double>(P,x-xB,y-yB,z-zB), result );
```

To evaluate its derivative, i. e., its Jacobian matrix of dimension $\mathbb{R}^{3\times 3}$, you would then use the following code and replace the regular harmonics `R` with their gradients `dR`:

```
double jacobian[9];
dot( L, harmonics::dR<double>(P,x-xB,y-yB,z-zB), jacobian );
```

Note that calling `harmonics::R` or `harmonics::dR` requires you to `#include` the `solidfmm/harmonics.hpp` header, which is described in Section 3.3.

3.2.5 Creating Expansions using `fmadd`

We recall the P2M and P2L operations from Chapter 2. Given a set of 'particles' with location $\mathbf{x}_i$ and mass m_i, $i = 0, \dots, N-1$, we desire to compute multipole or local expansions around respective centres $\mathbf{x}_A$ and $\mathbf{x}_B$:

$$M_n^m = \sum_{i=0}^{N-1} m_i R_n^m(\mathbf{x}_i - \mathbf{x}_A), \qquad L_n^m = \sum_{i=0}^{N-1} m_i S_n^m(\mathbf{x}_i - \mathbf{x}_B). \tag{3.2}$$

This operation thus always takes multiples of a `solid` containing the values of the regular or singular harmonics, and accumulates them into a `solid` that contains the expansion coefficients. There also is support for vector-valued expansions:

```
void fmadd( const float   fac, const solid<float > &A, solid<float > &B );
void fmadd( const float  *fac, const solid<float > &A, solid<float > &B );
void fmadd( const double  fac, const solid<double> &A, solid<double> &B );
void fmadd( const double *fac, const solid<double> &A, solid<double> &B );
```

In all of these operations one must have both `A.dimension() == B.dimension()` and `A.order() == B.order()`, because it is not clear how to meaningfully define those operations otherwise. These functions throw exceptions of type `std::logic_error` if one of these conditions is violated. Again, the strong exception guarantee is given: in case an exception is thrown, the function's arguments are not changed.

The functions taking a *value* `fac` essentially perform `B += fac*A`; but note that `solidfmm` does not implement `operator+=`. The operations taking a *pointer* `fac` assume that it points to an array of size `A.dimension() == B.dimension()`; the d'th component is then scaled by `fac[d]`.

We are still not sure how to devise a more flexible interface, we are open for suggestions!

3.3 `solidfmm/harmonics.hpp`

This header contains the function declations for evaluating the regular and singular harmonics, as well as their gradients, as they are defined in Section 2.3. The interface should be pretty self-explanatory:

```
namespace harmonics
{

template <typename real> solid<real>  R( size_t P, real x, real y, real z );
template <typename real> solid<real>  S( size_t P, real x, real y, real z );
template <typename real> solid<real> dR( size_t P, real x, real y, real z );
template <typename real> solid<real> dS( size_t P, real x, real y, real z );

}
```

The parameter `P` specifies the order up to which you wish to evaluate the harmonics. The parameters `x`,`y`, and `z` are the respective components of the position vector $\mathbf{x} = (x, y, z)^\top \in \mathbb{R}^3$ at which you want to evaluate the harmonic functions. Only `double` and `float` are supported for the template parameter `real`. These functions may throw exceptions of type `std::bad_alloc`.

3.4 solidfmm/handles.hpp

The handle classes encapsulate resources that are implementation details but are needed to carry out the single-threaded versions of the translation operations M2M, M2L, and L2L. You do not need these if you use the `multithreaded_translator` as described in Section 1.3.4.

There are two kinds of handles:

1. `operator_handle`. Only one of these handles is needed, it can and *should be shared* among threads. It is a wrapper to an implementation detail, which essentially stores the so-called Wigner matrices, which are needed for the efficient implementation of the translation operators. This class also automatically determines the available CPU features at runtime and chooses the most efficient implementation available.

2. `buffer_handle`. *Each thread needs its own handle.* These handles need to be created for a given `operator_handle`. As the name implies, a `buffer_handle` encapsulates buffers which a thread can use as its workspace when performing translations.

Handles can be copied, default constructed, and also support assignment operations. In a typical scenario, however, a user would create these handles *only once* at program start-up and then pass them around by reference. The relevant parts of the user-interface are as follows:

```
template <typename real>
class operator_handle
{
public:
        // …
    operator_handle( size_t P );
    buffer_handle<real> make_buffer() const;
        // …
};

template <typename real>
class buffer_handle
{
public:
        // …
    buffer_handle( const operator_data<real> &op );
        // …
};
```

In other words, a `buffer_handle` needs to be created specifically for a particular `operator_handle`. This can for example be achieved as follows:

```
// The maximum number of concurrent threads you are going to use.
// For example, on a CPU with 8 cores:
const size_t nthreads { 8 };

// Only one such object is necessary; it can and should be shared among threads.
// For exmaple, if you never need expansion orders beyond P = 30:
operator_handle<double> op  { 30 };

// Each thread needs its own buffer; it must not be shared.
std::vector<buffer_handle<double>> buffers(nthreads,op.make_buffer());
```

These routines may throw exceptions of type `std::bad_alloc`. Again, it is illegal to use other types than `double` and `float` for the template parameter `real`.

The translation operations make use of faculties up to $2P - 2$. For this reason, for `float` the maximum supported order is 18: for higher orders the number $(2P - 2)!$ is out of the range of this type. For double precision calculations the corresponding limit is 86. In both cases, the accuracies achieved by expansions of such high order are usually already exceeding the precision of the respective floating point numbers. In this case the constructors throw exceptions of type `std::overflow_error`. Passing $P = 0$ will case an exception of type `std::logic_error` to be thrown.

3.5 solidfmm/translations.hpp

This header contains methods for carrying out the translation operations M2M, M2L, and L2L. The mathematical background and details of these operations are discussed in Chapter 2. The `translation_info` and `multithreaded_translator` classes were already described in Section 1.3.4. Additionally this header contains the single threaded functions. To use these, you will need to create handles as described in Section 3.4.

3.5.1 Single-threaded Translation Routines

```
void           m2l( const operator_data<double> &op,
                          threadlocal_buffer<double> &buf,
                    const translation_info<double> *begin,
                    const translation_info<double> *end );

void m2l_unchecked( const operator_data<double> &op,
                          threadlocal_buffer<double> &buf,
                    const translation_info<double> *begin,
                    const translation_info<double> *end ) noexcept;

// Analogous for m2m, l2l, and single precision (float).
```

Just like the `multithreaded_translator`, these operations also support vector-valued `solids`. In this case, the translations are carried out component-wise along the specified shift vector. For this to work, the target `solids` must match the dimension of the input expansion.

The functions come in two variants. The default variants perform some simple runtime checks to ensure that the operations are well-defined. First of all, they check if the `operator_handle` and `buffer_handle` are compatible with each other. They then check that none of the coefficients exceed the maximum order P of the supplied handles. In the case of vector-valued coefficients, they check that dimensions of source and target expansions match. If one of these checks fails, an exception is thrown and the arguments are left unchanged (strong exception guarantee). In summary, the following exceptions can be thrown:

- `std::logic_error` If input and output dimensions of the passed `solids` differ. This also is thrown if you pass a `buffer_handle` that was created for a different `operator_handle`.
- `std::out_of_range` if the order of one of the passed `solids` exceeds the order of the given `operator_handle`.

The unchecked variants omit these checks, never throw and exception and lead to *undefined behaviour* when invalid data is supplied.

The types `operator_data` and `threadlocal_buffer` are the implementation details that are encapsulated by the handle classes in Section 3.4. Thus, a user should pass the corresponding handles instead, and *never* use the internal classes `operator_data` and `threadlocal_buffer` directly.

Just like the `multithreaded_translator` class, these functions *add* and accumulate the translation results into the corresponding targets:

$$\text{target} \quad \texttt{+=} \quad \text{translate}(\text{source}, x, y, z).$$

These functions do not create any threads, they operation is carried out by the calling thread of execution. You may call `m2m`, `l2l`, and `m2l` simultaneously from several threads. In this case, each thread needs its own buffer. You cannot have different threads write to the same output, this will cause a race condition and undefined results.

3.5.2 Single-threaded Methods using the Classic Interface

The classic interface only comes in the ‘checked’ variant and behaves identically to the interface using the `translation_info` class described in the previous subsection.

```
void m2m( const operator_data<double> &op, threadlocal_buffer<double> &buf,
          size_t howmany, const solid<double> *const *const Min,
                                solid<double> *const *const Mout,
          const double *x, const double *y, const double *z );

void m2l( const operator_data<double> &op, threadlocal_buffer<double> &buf,
          size_t howmany, const solid<double> *const *const M,
                                solid<double> *const *const L,
          const double *x, const double *y, const double *z );


void l2l( const operator_data<double> &op, threadlocal_buffer<double> &buf,
          size_t howmany, const solid<double> *const *const Lin,
                                solid<double> *const *const Lout,
          const double *x, const double *y, const double *z );
```

and the corresponding functions for single precision calculations using `float`.

Again, the types `operator_data` and `threadlocal_buffer` are the implementation details that are encapsulated by the handle classes in Section 3.4. Thus, a user should pass the corresponding handles instead, and *never* use the internal classes `operator_data` and `threadlocal_buffer` directly.

Assume we were given the following data, initialised somewhere else:

```
// Number of translations you want to carry out.
size_t howmany;

// Arrays of size howmany, containing *pointers* to coefficients.
solidfmm::solid<double> *input [howmany];
solidfmm::solid<double> *output[howmany];

// Shift vectors for the translations. That is, if the input coefficients
// represent expansions about a point (xA,yA,zA), and the output coefficients
// should be around another centre (xB,yB,zB), the shift-vectors would be
// (xB-xA,yB-yA,zB-zA).
double xshift[howmany];
double yshift[howmany];
double zshift[howmany];
```

We also assume that you already have created operator and buffer handles as described in Section 3.4. To perform the actual translation, you just need to call M2L, or, respectively m2m and l2l, as follows:

```
solidfmm::m2l(op_handle,buf_handle,howmany,input,output,xshift,yshift,zshift);
```

Part II

Developer Guide

Chapter 4

Microkernels

At their core, the translations M2M, M2L, and L2L break down to certain copy operations and dense matrix–matrix products. However, every CPU architecture is different, hence each requires its own specific, optimised implementation of these operations to obtain maximum performance. This leads to the question: 'How can we write *one* library that achieves high performance on *many* platforms?'

To answer this question, `solidfmm` follows the approach taken the BLIS developers and uses the concept of microkernels.[14],[15] While each CPU requires its own implementation, microkernels encapsulate the key operations behind a unified interface. The 'driver code' then can remain machine independent, while the machine specific code is reduced to a well-defined, isolated, small set of functions. Due to this localisation, the library can be quickly adapted to new architectures.

In C++, the most natural choice to represent such an interface is an abstract base class gives this interface in the form of `virtual` functions. For each new CPU architecture optimised implementations can then be provided by deriving from this base class and overriding the generic default routines. The driver routines only make use of the base class, and through run-time polymorphism the optimised, machine specific routines get called.

In this chapter we describe `solidfmm`'s microkernel interface. We specify the functionality that these functions must implement, how to include such a new microkernel into `solidfmm`, and how to test that it is working. How to achieve maximum performance on your computer is, of course, machine dependent. However, many routines are similar to general matrix–matrix products. As a starting point, we thus suggest to take a look at the BLIS,[16] which comes with highly efficient microkernels for a variety of platforms. These can serve as an inspiration when writing your own microkernels for `solidfmm`. Their authors furthermore created an excellent lecture series in which they share their expertise.[15]

4.1 Overview

The `microkernel` class is declared in its own header file:

```
#include <solidfmm/microkernel.hpp>
```

Its interface is given in the following listing. As usual, the only permissible values for the template parameter `real` are `float` and `double`.

```
template <typename real>
class microkernel
{
public:
    const size_t rows;
    const size_t cols;
    const size_t alignment;

    microkernel( size_t p_rows, size_t p_cols, size_t p_alignment ) noexcept:
    rows { p_rows }, cols { p_cols }, alignment { p_alignment } {}

    virtual ~microkernel() = default;

    virtual void euler( const real *x,   const real *y, const real *z,
                              real *r,         real *rinv,
                              real *cos_alpha, real *sin_alpha,
                              real *cos_beta,  real *sin_beta,
                        size_t k ) const noexcept = 0;

    virtual void rotscale( const real *cos, const real *sin, const real *scale,
                           const real *real_in,  const real *imag_in,
                                 real *real_out,       real *imag_out,
                           size_t k, bool forward ) const noexcept = 0;

    virtual void swap( const real *mat, const real *in,
                       real *out, size_t k, bool pattern ) const noexcept = 0;

    virtual void zm2l( const real *mat, const real *in,
                       real *out, size_t k, bool pattern ) const noexcept = 0;

    virtual void zm2m( const real *mat, const real *in,
                       real *out, size_t k ) const noexcept = 0;

    virtual void swap2trans_buf( const real  *real_in,  const real  *imag_in,
                                       real **real_out,       real **imag_out,
                                 size_t n ) const noexcept = 0;

    virtual void trans2swap_buf( const real *const *const real_in,
                                 const real *const *const imag_in,
                                       real *real_out, real *imag_out,
                                 size_t n, size_t Pmax ) const noexcept = 0;

    virtual void solid2buf( const real *const *solids,
                  const real *const  zeros, const size_t *P,
                  real *real_out, real *imag_out, size_t n ) const noexcept = 0;

    virtual void buf2solid( const real *real_in, const real *imag_in,
                            real **solids, real *trash,
                            const size_t *P, size_t n ) const noexcept = 0;
};
```

Listing 4.1: Interface of the `microkernel` class.

4.1.1 Data Members

Block Size. As described before, in the course of a translation operation, certain matrix–matrix products $C = AB$ need to be computed. `solidfmm` computes such products in a block-wise fashion: the matrix C is sub-divided into blocks of size `rows` × `cols` and then calls the necessary microkernel routines. Together these two parameters are called *block size*. The block size should be chosen according to the following guidelines:

- Even number of rows. The parameter `rows` *must* be a multiple of two.
- Register space. The block-size should be as large as possible such that it can still fit into the processor's registers, and such that enough spare registers remain for forming the matrix–matrix products. Ideally you use exactly all available registers and leave no un-used registers behind.
- Ratio. The parameters should fulfil `rows` ≈ `cols`, yielding approximately square blocks. This ensures optimal data reuse and avoids unnecessary memory transfers.

For the rest of this chapter we will write μ ≔ `rows` and ν ≔ `cols`.

Alignment. The `alignment` parameter causes `solidfmm` to ensure that the first entry of the matrices B and C (but *not* A) will be aligned as specified. A k byte alignment means that the memory address is a multiple of k. For example, 32 byte alignment causes the addresses of the respective first entries of B and C to be divisible by 32. Many processors offer extra fast instructions for aligned memory access. Use this parameter to be able to make use of these optimisations. There are certain restrictions that this parameter must fulfil:

- It must be a power of two, i. e., `alignment` $= 2^l$ for some integer $l \in \mathbb{N}_0$.
- On POSIX systems, it must be a power of two multiple of `sizeof(void*)`, i. e., on these systems there must exist some $k \in \mathbb{N}$ such that

  $$\texttt{alignment} = 2^k \times \texttt{sizeof(void*)}.$$

- The size of one row needs to be a fixed multiple of `alignment`:

  $$\texttt{sizeof(real)} \times \texttt{cols} = 0 \pmod{\texttt{alignment}},$$

  or, equivalently, in C++ syntax, one must have:

  ```
  (sizeof(real)*cols) % alignment == 0
  ```

  This ensures that the beginning of every row of B and C has the same alignment.

4.1.2 Function Members

There are three kinds of routines in a `microkernel`:

- Copy routines: `solid2buf`, `buf2solid`, `swap2trans_buf`, and `trans2swap_buf`. These operations copy and rearrange data between different locations.
- Matrix–matrix products: `swap`, `m2m`, and `m2l`. These routines carry out specialised matrix–matrix products $C = AB$, where C is of dimension `rows` × `cols`. The remaining dimension of these matrices is called `k` and given as a parameter at run-time.
- Other: `euler` and `rotscale` will be described separately.

We will describe each of these routines in greater detail below.

4.2 Copy Routines

4.2.1 `solid2buf()`

```
virtual void solid2buf( const real *const *solids,
                        const real *const  zeros, const size_t *P,
                        real *real_out, real *imag_out, size_t n ) const noexcept;
```

This method copies the line n of given solids into buffers for further processing.
We are given ν *scalar* solids $C_n^m[0]$, $C_n^m[1]$, …, $C_n^m[\nu-1]$ of respective orders $P[0], P[1], \ldots, P[\nu-1]$. The parameter `solids` is an array of pointers to these solids. Each of these solids is stored as described in Figure 3.1. In other words, to access $\Re C_n^m[k]$ for some n, m, and k you would write:

```
solids[k][ (n+0)*(n+1) + m ]
```

Here we assumed $n, m, k \in \mathbb{N}_0$ and that $k < \nu$, $n < P[k]$, $m \leq n$. Whenever $n \geq P[k]$ it is implicitly assumed that $C_n^m[k] = 0$. Thus, more correctly, to access the value of, e. g., $\Im C_n^m[k]$, the complete code snippet would be:

```
real value = (n<P[k]) ? solids[k][ (n+1)*(n+1) + m ] : 0;
```

In the method `solid2buf()` the value of n is fixed and given as a parameter. The task is now to copy row n of each solid into the matrices `real_out` and `imag_out`, that respectively hold the real and imaginary parts of the solids. These matrices are stored in *row-major* order and look as follows:

$$\begin{pmatrix} \Re C_n^0[0] & \Re C_n^0[1] & \cdots & \Re C_n^0[\nu-1] \\ \Re C_n^1[0] & \Re C_n^1[1] & \cdots & \Re C_n^1[\nu-1] \\ \vdots & \vdots & \ddots & \vdots \\ \Re C_n^n[0] & \Re C_n^n[1] & \cdots & \Re C_n^n[\nu-1] \end{pmatrix} \begin{pmatrix} \Im C_n^0[0] & \Im C_n^0[1] & \cdots & \Im C_n^0[\nu-1] \\ \Im C_n^1[0] & \Im C_n^1[1] & \cdots & \Im C_n^1[\nu-1] \\ \vdots & \vdots & \ddots & \vdots \\ \Im C_n^n[0] & \Im C_n^n[1] & \cdots & \Im C_n^n[\nu-1] \end{pmatrix} \tag{4.1}$$

Thus, the behaviour of this function is equivalent to the following snippet:

```
for ( size_t m = 0; m <= n;    ++m )
for ( size_t k = 0; k <= cols; ++k )
{
    real_out[ m*cols + k ] = (n<P[k]) ? solids[k][ (n+0)*(n+1) + m ] : 0;
    imag_out[ m*cols + k ] = (n<P[k]) ? solids[k][ (n+1)*(n+1) + m ] : 0;
}
```

The parameter `zeros` points to a memory location containing zero values, that may be used to avoid repeatedly checking if we have `n<P[k]`. This can be done by creating a local pointer array, leading to the following solution of the task:

```
const real* tmp_solids[ cols ];
for ( size_t k = 0; k < cols; ++k )
{
    if ( n < P[k] )
        tmp_solids[ k ] = const_cast<real*>(solids[k]);
    else
        tmp_solids[ k ] = const_cast<real*>(zeros);
}

for ( size_t m = 0; m <= n;    ++m )
for ( size_t k = 0; k <= cols; ++k )
{
    real_out[ m*cols + k ] = tmp_solids[k][ (n+0)*(n+1) + m ];
    imag_out[ m*cols + k ] = tmp_solids[k][ (n+1)*(n+1) + m ];
}
```

This way the condition `n < P[k]` only needs to be checked once per solid.

The matrices `real_out` and `imag_out` are aligned according to the `alignment` member of the microkernel object. The solids will usually be unaligned. Additionally, the solids may alias, i. e., `solid[i]` may point to the same location as `solid[j]`. Keep this in mind when trying to optimise this operation.

4.2.2 buf2solid()

```
virtual void buf2solid( const real *real_in, const real *imag_in,
                        real **solids, real *trash, const size_t *P, size_t n )
                        const noexcept;
```

In essence, this is the inverse to the routine `solid2buf()`, which we described in the previous subsection. In other words, the data from the given buffers is copied to the target solids. If the order of the target solid is below `n`, the data gets discarded. The `trash` parameter can be used to write data in this case. This prevents frequent rechecking if we have `P[k]<n`, similar to the `zeros` parameter from `solid2buf()`.

However, unlike in `solid2buf()`, the data is *added* to the output solids. This enables the users to accumulate several input expansions into a single output expansion. It is thus crucial to keep in mind that the pointers from the `solids` array *may and often do alias*. Thus, after adding to `solids[i]` for some i, it is important to keep in mind that `solid[j]` might also have been changed, as we might have `solid[i]==solid[j]` even when $i \neq j$. In an assembly language implementation this means that `solid[i+1]` can only be read from memory *after writing* `solid[i]`. A simple default implementation in C++ looks as follows:

```
real* tmp_solids[ cols ];
for ( size_t k = 0; k < cols; ++k )
    tmp_solids[ k ] = ( n < P[k] ) ? solids[k] : trash;

for ( size_t m = 0; m <= n;    ++m )
for ( size_t k = 0; k <= cols; ++k )
{
    tmp_solids[k][ (n+0)*(n+1) + m ] += real_in[ m*cols + k ];
    tmp_solids[k][ (n+1)*(n+1) + m ] += imag_in[ m*cols + k ];
}
```

Note that both `solid2buf()` and `buf2solid()` are similar to transposing a matrix in memory. When creating high-performance implementations of this operation, you can try using tricks for matrix transposition to obtain higher performance. Also note that `real_in` and `imag_in` are aligned according to the `alignment` member, but the individual solids are usually not aligned.

4.2.3 `swap2trans_buf()`

```
virtual void swap2trans_buf( const real  *real_in,  const real  *imag_in,
                                   real **real_out,       real **imag_out,
                                   size_t n ) const noexcept;
```

There are two kinds of buffers in `solidfmm`: swap and translation buffers. Swap buffers hold *some row n* of ν solids $C[0], C[1], \dots, C[\nu-1]$, i. e., they are matrices of the shape

$$\begin{pmatrix} \Re C_n^0[0] & \Re C_n^0[1] & \dots & \Re C_n^0[\nu-1] \\ \Re C_n^1[0] & \Re C_n^1[1] & \dots & \Re C_n^1[\nu-1] \\ \Re C_n^2[0] & \Re C_n^2[1] & \dots & \Re C_n^2[\nu-1] \\ \vdots & \vdots & \ddots & \vdots \\ \Re C_n^n[0] & \Re C_n^n[1] & \dots & \Re C_n^n[\nu-1] \end{pmatrix} \quad \begin{pmatrix} \Im C_n^0[0] & \Im C_n^0[1] & \dots & \Im C_n^0[\nu-1] \\ \Im C_n^1[0] & \Im C_n^1[1] & \dots & \Im C_n^1[\nu-1] \\ \Im C_n^2[0] & \Im C_n^2[1] & \dots & \Im C_n^2[\nu-1] \\ \vdots & \vdots & \ddots & \vdots \\ \Im C_n^n[0] & \Im C_n^n[1] & \dots & \Im C_n^n[\nu-1] \end{pmatrix},$$

and are stored in row-major format. Conversely, a translation buffer stores *some column* m of ν different solids. For example, for a fixed m, the real-part of a translation buffer looks like follows:

$$\begin{pmatrix} \Re C_m^m[0] & \Re C_m^m[1] & \dots & \Re C_m^m[\nu-1] \\ \Re C_{m+1}^m[0] & \Re C_{m+1}^m[1] & \dots & \Re C_{m+1}^m[\nu-1] \\ \Re C_{m+2}^m[0] & \Re C_{m+2}^m[1] & \dots & \Re C_{m+2}^m[\nu-1] \\ \vdots & \vdots & \ddots & \vdots \end{pmatrix},$$

where the matrix is stored in row-major format and the imaginary parts are stored analogously.

This method takes the content of the swap buffers given as `real_in` and `imag_in` of the given row `n` and copies it to the corresponding positions in the translation buffers. A simple C++ implementation would look like follows.

```
for ( size_t m = 0; m ⩽  n;    ++m )
for ( size_t k = 0; k <  cols; ++k )
{
    real_out[m][ (n-m)*cols + k ] = real_in[ m*cols + k ];
    imag_out[m][ (n-m)*cols + k ] = imag_in[ m*cols + k ];
}
```

All of the buffers are aligned according to the `alignment` member. This enables the use of fast, aligned copy instructions on certain machines.

4.2.4 trans2swap_buf()

```
virtual void trans2swap_buf( const real *const *const real_in,
                             const real *const *const imag_in,
                                   real *real_out, real *imag_out,
                             size_t n, size_t Pmax ) const noexcept;
```

This is the inverse operation to `swap2trans_buf`, as described above. The parameter `Pmax` indicates that values $C_n^m[k]$ with $m \geq P_{\max}$ are considered to be zero. A simple C++ implementation could thus look as follows:

```
// First the entries where m < Pmax
for ( size_t m = 0; m < min(Pmax,n+1); ++m )
for ( size_t k = 0; k < cols;          ++k )
{
    real_out[ m*cols + k ] = real_in[m][ (n-m)*cols + k ];
    imag_out[ m*cols + k ] = imag_in[m][ (n-m)*cols + k ];
}

// Now the remaining entries, if any.
for ( size_t m = Pmax; m ⩽  n;    ++m )
for ( size_t k = 0;    k <  cols; ++k )
{
    real_out[ m*cols + k ] = 0;
    imag_out[ m*cols + k ] = 0;
}
```

4.3 Matrix–matrix Product Routines

The methods of this section compute matrix–matrix products $C = AB$, where $C \in \mathbb{R}^{\mu\times\nu}$, $A \in \mathbb{R}^{\mu\times k}$, and $B \in \mathbb{R}^{k\times\nu}$. For convenience we abbreviate μ ≔ `rows` and ν ≔ `cols`. We will denote the mathematical matrices with capital letters; pointers to the corresponding data will be denoted by denoted by lower case letters.

The matrices C and B are stored in row-major format, e. g., C_{ij} can be accessed via `c[i*cols + j]`, where $i = 0, \dots, \mu - 1$ and $j = 0, \dots, \nu - 1$. Similarly, B_{ij} can be accessed via `b[i*cols+j]`, where $i = 0, 1, \dots, k - 1$ and $j = 0, 1, \dots, \nu - 1$. Both `c` and `b` will be aligned according to the `alignment` class member.

The pointer `a` will usually not be aligned. The matrix A is stored in special formats, depending on the particular routine.

4.3.1 `swap()`

```
virtual void swap( const real *a, const real *b, real *c, size_t k,
                   bool pattern ) const noexcept;
```

This routine is called `swap`, because it is used in coordinate transformations that correspond to swapping the x- and z-axis. The matrix A here has one of two possible chequerboard patterns. Only half of A's entries are non-zero. Only these non-zero entries are stored in `a`, using a compressed column major format. In particular, if the `pattern` parameter is true, A's first entry is non-zero. For example, when $\mu = 6$, the matrix A then is given by:

$$A = \underbrace{\begin{pmatrix} a[0] & 0 & a[6] & 0 & \cdots \\ 0 & a[3] & 0 & a[9] & \cdots \\ a[1] & 0 & a[7] & 0 & \cdots \\ 0 & a[4] & 0 & a[10] & \cdots \\ a[2] & 0 & a[8] & 0 & \cdots \\ 0 & a[5] & 0 & a[11] & \cdots \end{pmatrix}}_{k \text{ columns}} \qquad \text{if } \texttt{pattern==true}. \tag{4.2}$$

Otherwise, if `pattern==false`, the matrix A is given by:

$$A = \underbrace{\begin{pmatrix} 0 & a[3] & 0 & a[9] & \cdots \\ a[0] & 0 & a[6] & 0 & \cdots \\ 0 & a[4] & 0 & a[10] & \cdots \\ a[1] & 0 & a[7] & 0 & \cdots \\ 0 & a[5] & 0 & a[11] & \cdots \\ a[2] & 0 & a[8] & 0 & \cdots \end{pmatrix}}_{k \text{ columns}} \quad \text{if } \texttt{pattern==false}. \tag{4.3}$$

4.3.2 `zm2l()`

```
virtual void zm2l( const real *a, const real *b, real *c, size_t k,
                   bool pattern ) const noexcept;
```

This operation is needed when performing an M2L translation along the z-axis. Actually, two operations are carried out. First, the matrix–matrix product $C = AB$ is formed, where $A \in \mathbb{R}^{\mu\times k}$ is given as follows:

$$A = \underbrace{\begin{pmatrix} a[0] & a[1] & a[2] & a[3] & \cdots \\ a[1] & a[2] & a[3] & a[4] & \cdots \\ a[2] & a[3] & a[4] & a[5] & \cdots \\ a[3] & a[4] & a[5] & a[6] & \cdots \\ \vdots & \vdots & \vdots & \vdots & \\ a[\mu-1] & a[\mu] & a[\mu+1] & a[\mu+2] & \cdots \end{pmatrix}}_{k \text{ columns}}. \tag{4.4}$$

In other words, the value of A_{ij} can be accessed via `a[i+j]`, where $i = 0, 1, \dots, \mu - 1$ and $j = 0, 1, \dots, k - 1$.

Once the product C has been formed, we apply a sign pattern to the result:

- If `pattern==true`: rows 0, 2, 4, …, μ-2 of the result matrix C are multiplied by (-1).
- If `pattern==false`: rows 1, 3, 5, …, μ-1 of the result matrix C are multiplied by (-1).

4.3.3 `zm2m()`

```
virtual void zm2m( const real *a, const real *b, real *c, size_t k ) const noexcept;
```

This operation is needed when performing M2M or L2L translations along the z-axis.

It carries out a matrix–matrix product $C = AB$, where $A \in \mathbb{R}^{\mu\times k}$ is given as follows:

$$A = \underbrace{\begin{pmatrix} a[\mu-1] & a[\mu] & a[\mu+1] & a[\mu+2] & \cdots \\ a[\mu-2] & a[\mu-1] & a[\mu] & a[\mu+1] & \cdots \\ \vdots & \vdots & \vdots & \vdots & \\ a[1] & a[2] & a[3] & a[4] & \cdots \\ a[0] & a[1] & a[2] & a[3] & \cdots \end{pmatrix}}_{k \text{ columns}}. \tag{4.5}$$

In other words, the matrix A corresponds to that of `zm2l()`, but with the order of the rows reversed.

4.4 euler and rotscale

4.4.1 euler()

```
virtual void euler( const real *x,   const real *y, const real *z,
                          real *r,         real *rinv,
                          real *cos_alpha, real *sin_alpha,
                          real *cos_beta,  real *sin_beta,
                    size_t k ) const noexcept;
```

Given ν shift vectors with coordinates $(x[i], y[i], z[i])^\top$, $i = 0, \dots, \nu-1$, this method computes data that is related to the Euler angles of a certain coordinate transform.

This method fills the matrix `r` powers of the lengths of these vectors and is stored in row-major format:

$$\mathtt{r} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ r[0] & r[1] & \dots & r[\nu-1] \\ r[0]^2 & r[1]^2 & \dots & r[\nu-1]^2 \\ \vdots & \vdots & \ddots & \vdots \\ r[0]^{k-1} & r[1]^{k-1} & \dots & r[\nu-1]^{k-1} \end{pmatrix},$$

where $r[i] := \sqrt{x[i]^2 + y[i]^2 + z[i]^2}$.

The matrix `rinv` is filled with inverse powers thereof:

$$\mathtt{rinv} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ r[0]^{-1} & r[1]^{-1} & \dots & r[\nu-1]^{-1} \\ r[0]^{-2} & r[1]^{-2} & \dots & r[\nu-1]^{-2} \\ \vdots & \vdots & \ddots & \vdots \\ r[0]^{-(k-1)} & r[1]^{-(k-1)} & \dots & r[\nu-1]^{-(k-1)} \end{pmatrix}.$$

The matrices `cos_alpha` and `sin_alpha` respectively store the real and imaginary parts powers of $e^{I\alpha[i]}$, $i = 1, \dots, \nu-1$, where $I^2 = -1$ is the imaginary unit and:

$$\Re e^{I\alpha[i]} := \frac{y[i]}{\sqrt{x[i]^2 + y[i]^2}}, \quad \Im e^{I\alpha[i]} := \frac{x[i]}{\sqrt{x[i]^2 + y[i]^2}},$$

with one special exception: if $\sqrt{x[i]^2 + y[i]^2} = 0$, one sets $\Re e^{I\alpha[i]} := 1$ and $\Im e^{I\alpha[i]} := 0$. Powers of these numbers can be computed using the complex product of two numbers alone, no trigonometry is needed. However, as $e^{Ik\alpha[i]} = \cos(k\alpha[i]) + I\sin(k\alpha[i])$, the matrices `cos_alpha` and `sin_alpha` shall be filled by `euler()` with the respective real and imaginary parts of $e^{Ik\alpha[i]}$ as follows:

$$\texttt{cos\_alpha} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \cos(\alpha[0]) & \cos(\alpha[1]) & \dots & \cos(\alpha[\nu-1]) \\ \cos(2\alpha[0]) & \cos(2\alpha[1]) & \dots & \cos(2\alpha[\nu-1]) \\ \cos(3\alpha[0]) & \cos(3\alpha[1]) & \dots & \cos(3\alpha[\nu-1]) \\ \vdots & \vdots & \ddots & \vdots \\ \cos(k\alpha[0]) & \cos(k\alpha[1]) & \dots & \cos(k\alpha[\nu-1]) \end{pmatrix},$$

$$\texttt{sin\_alpha} = \begin{pmatrix} 0 & 0 & \dots & 0 \\ \sin(\alpha[0]) & \sin(\alpha[1]) & \dots & \sin(\alpha[\nu-1]) \\ \sin(2\alpha[0]) & \sin(2\alpha[1]) & \dots & \sin(2\alpha[\nu-1]) \\ \sin(3\alpha[0]) & \sin(3\alpha[1]) & \dots & \sin(3\alpha[\nu-1]) \\ \vdots & \vdots & \ddots & \vdots \\ \sin(k\alpha[0]) & \sin(k\alpha[1]) & \dots & \sin(k\alpha[\nu-1]) \end{pmatrix}.$$

Finally, the matrices `cos_beta` and `sin_beta` are defined analogously, where $e^{I\beta[i]}$ is given as follows:

$$\Re e^{I\beta[i]} := \frac{z[i]}{\sqrt{x[i]^2 + y[i]^2 + z[i]^2}}, \quad \Im e^{I\beta[i]} := -\frac{\sqrt{x[i]^2 + y[i]^2}}{\sqrt{x[i]^2 + y[i]^2 + z[i]^2}},$$

and, if $x[i] = y[i] = z[i] = 0$: $\Re e^{I\beta[i]} := 1$, $\Im e^{I\beta[i]} := 0$.

4.4.2 rotscale()

```
virtual void rotscale( const real *cos,      const real *sin, const real *scale,
                       const real *real_in,  const real *imag_in,
                             real *real_out,       real *imag_out,
                       size_t k, bool forward ) const noexcept;
```

This operation performs an element-wise, complex multiplication $C = A \odot B$ of matrices of the same size $\mathbb{C}^{k\times\nu}$, and stores the resulting real and imaginary parts of the matrix C respectively in `real_out` an `imag_out`.

The matrices of `cos` and `sin` contain the real and imaginary parts of powers of $e^{I\gamma}$, where $\gamma \in \{\alpha, \beta\}$, and α and β are defined in the method `euler()` above. If `forward`=`true`, then the products are formed with powers of $e^{I\gamma}$, otherwise with its conjugate $e^{-I\gamma}$.

Finally, each row of the result gets scaled element-wise with the values in `scale`. A simple C++ implementation of this method is as follows:

```
if ( forward )
{
    for ( size_t i = 0; i < k;    ++i )
    for ( size_t l = 0; l < cols; ++l )
    {
        real c   = cos    [ i*cols + l ];
        real s   = sin    [ i*cols + l ];
        real re  = real_in[ i*cols + l ];
        real im  = imag_in[ i*cols + l ];
        real fac = scale  [          l ];

        real_out[ i*cols + l ] = fac*(c*re - s*im);
        imag_out[ i*cols + l ] = fac*(s*re + c*im);
    }
}
else
{
    for ( size_t i = 0; i < k;    ++i )
    for ( size_t l = 0; l < cols; ++l )
    {
        real c   = cos    [ i*cols + l ];
        real s   = sin    [ i*cols + l ];
        real re  = real_in[ i*cols + l ];
        real im  = imag_in[ i*cols + l ];
        real fac = scale  [          l ];

        real_out[ i*cols + l ] = fac*( c*re + s*im);
        imag_out[ i*cols + l ] = fac*(-s*re + c*im);
    }
}
```

All matrices are aligned according to the `alignment` member, the `scale` array usually is not aligned.

4.5 Writing your own Microkernel

In this section we suppose that you want to write optimised routines for a fictional new processor called ‘aicpu’, whose name is based on a currently severely overhyped research topic.

4.5.1 Preparation

Before you start writing your own microkernel, you should be familiar with the basics of high performance dense linear algebra. A very detailed guide is given on the LAFF website.[15]

You will need to determine a suitable block size for your processor. The computationally most expensive routines of solidfmm are specialised dense matrix–matrix products; it is these routines which should determine the block size of your microkernel. For many current architectures, you can find microkernels for the *general* matrix–matrix multiplication (GEMM) in the BLIS library.[16] These are high-quality kernels and form a good starting point for the specialised kernels of solidfmm. Usually, their block size already fulfils the requirements described in Section 4.1.1.

Naturally, you should familiarise yourself with the programming guides of your processor manufacturer. These usually contain detailed information on how to achieve high performance and on important details such as alignment requirements and the sizes of caches and cache lines.

Finally, to develop solidfmm, on top of a current C++ compiler, you will need the GNU Autotools autoconf, automake, libtool, and the autoconf macro library.

4.5.2 Creating Files

You need to create two new files in the solidfmm subfolder of the source code distribution:

- microkernel_aicpu.hpp, containing the class defintition of your microkernel(s). If you want to support both single and double precision, i. e., float and double, this header should contain two classes, one for each floating point type.
- microkernel_aicpu.cpp, containing the microkernel's actual implementation.

For example, when implementing both single and double precision, the header file microkernel_aicpu.hpp could look as follows.

```
#include <solidfmm/microkernel.hpp>

namespace solidfmm
{

class microkernel_aicpu_float: public microkernel<float>
{
public:
    microkernel_aicpu_float();
    virtual ~microkernel_aicpu_float() = default;

    virtual void euler( const float *x,   const float *y, const float *z,
                              float *r,         float *rinv,
                              float *cos_alpha, float *sin_alpha,
                              float *cos_beta,  float *sin_beta,
                        size_t k ) const noexcept override;

    // Similarly, override all the other members of the microkernel base class.

    // Consider adding a static routine that checks at runtime whether the
    // kernel can run on the currently used CPU.
    static bool available() noexcept;
};

class microkernel_aicpu_double: public microkernel<double>
{
public:
    microkernel_aicpu_double();
    virtual ~microkernel_aicpu_double() = default;

    virtual void euler( const double *x,   const double *y, const double *z,
                              double *r,         double *rinv,
                              double *cos_alpha, double *sin_alpha,
                              double *cos_beta,  double *sin_beta,
                        size_t k ) const noexcept override;

    // Similarly, override all the other members of the microkernel base class.

    // Consider adding a static routine that checks at runtime whether the
    // kernel can run on the currently used CPU.
    static bool available() noexcept;
};

}
```

On certain platforms, it makes sense to add static routines called `available()` that perform runtime checks whether the kernel is available on the currently running CPU. The constructors should call this routine and throw an exception if the currently running CPU is not compatible with the 'aicpu'.

4.5.3 Writing the Actual Code

Next you need to write the actual implementation of your microkernel in the freshly created `microkernel_aicpu.cpp` file. For this, we refer to the specification of the individual routines in the previous sections.

It is a good strategy to start with the generic implementations from `microkernel_generic.cpp` and start replacing the individual routines with optimised versions one by one. It is also worth looking at or the other optimised implementations for an inspiration.

4.5.4 Modifying the Build System

Once you have finished writing your code, you need to inform the build system about its existence. For this you need to edit the file `Makefile.am`. Look for the line starting with `noinst_HEADERS` and add the header `solidfmm/microkernel_aicpu.hpp` to the list. Next, add the file `solidfmm/microkernel_aicpu.cpp` to the line beginning with `libsolidfmm_la_SOURCES`. Lastly, you should increase the first and last digits of the 'version-info' of the library. The result should look similar to the following.

```
# Do not touch these lines.
ACLOCAL_AMFLAGS = -I m4
EXTRA_DIST = doc/README doc/solidfmm.tex doc/literature.bib doc/solidfmm.pdf   \
             m4/ax_cxx_compile_stdcxx.m4 m4/ax_pthread.m4 m4/pkg.m4            \
             kirchhart_pubkey.gpg solidfmm.pc.in

pkgconfig_DATA = solidfmm.pc

lib_LTLIBRARIES = solidfmm/libsolidfmm.la

pkginclude_HEADERS = solidfmm/solid.hpp solidfmm/harmonics.hpp                 \
    solidfmm/translations.hpp solidfmm/handles.hpp solidfmm/sphere.hpp         \

# Add your header here
noinst_HEADERS = solidfmm/random.hpp solidfmm/stopwatch.hpp                    \
    solidfmm/swap_matrix.hpp solidfmm/microkernel.hpp                          \
    solidfmm/microkernel_avx.hpp solidfmm/microkernel_avx512f.hpp              \
    solidfmm/microkernel_generic.hpp solidfmm/microkernel_test.hpp             \
    solidfmm/microkernel_armv8a.hpp solidfmm/operator_data.hpp                 \
    solidfmm/threadlocal_buffer.hpp solidfmm/microkernel_aicpu.hpp

# Add your implementation file here
solidfmm_libsolidfmm_la_SOURCES = solidfmm/solid.cpp solidfmm/harmonics.cpp    \
    solidfmm/handles.cpp solidfmm/sphere.cpp  solidfmm/swap_matrix.cpp         \
    solidfmm/operator_data.cpp solidfmm/threadlocal_buffer.cpp                 \
    solidfmm/microkernel.cpp solidfmm/microkernel_avx.cpp                      \
    solidfmm/microkernel_avx512f.cpp solidfmm/microkernel_generic.cpp          \
    solidfmm/microkernel_armv8a.cpp solidfmm/microkernel_test.cpp              \
    solidfmm/translations.cpp solidfmm/multithreaded_translator.cpp            \
    solidfmm/microkernel_aicpu.cpp

# Increase version, e.g., from 4:0:4 to 5:0:5
solidfmm_libsolidfmm_la_LDFLAGS = -version-info 5:0:5

# No need to modify the file below here.
```

More information about the version info can be found in the `libtool` documentation.

Once you have completed these steps, you can compile your code by entering `make` on the command shell. The build system will compile your code and perform the necessary steps to link the new library.

4.5.5 Testing and Benchmarking

Once you have managed to get rid of all compile time errors, it is time to test whether your kernel behaves as expected. For this purpose a testing programme was written that can be found in the `tests` subfolder. Edit the code on the top of the file `tests/test_microkernel.cpp` to compare your implementation against a generic, slow reference implementation. Running the programme will check each routine on a set of random data and test whether it produces the expected results.

You can then finally modify the file `tests/benchmark_microkernel.cpp`. After compiling, you can run the corresponding programme to benchmark your microkernel against a generic implementation.

4.5.6 Enabling the Microkernel for the Library

The final step is to enable the microkernel for the library. For this, you need to edit the methods `get_microkernel<float>()` and `get_microkernel<double>()` in the file `solidfmm/microkernel.cpp`. Edit these functions such that they return your microkernels on the corresponding devices. How to detect a device type at compile- or run-time depends on the specific machine and goes beyond the scope of this book.

Appendices

Appendix A

GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each

Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/licenses/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with … Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B

GNU General Public License

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

   A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

   The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

   The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met.

This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

   "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

   When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

   Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

   a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
   b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
   c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as

different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

End of Terms and Conditions

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

Bibliography

[1] T. Authors. *The Hardware Locality Project*. Computer Software. URL: https://www.open-mpi.org/projects/hwloc/.

[2] The GnuPG Project. *GNU Privacy Guard*. Computer software. URL: https://gnupg.org/.

[3] The GNU Project. *GNU Tar*. Computer software. URL: https://www.gnu.org/software/tar/.

[4] Tukaani Developers. *XZ Utils*. Computer software. URL: https://tukaani.org/xz/.

[5] J. Calcote. *Autotools. A Practitioner's Guide to GNU Autoconf, Automake, and Libtool*. San Francisco: no starch press, 2010. ISBN: 1593272065.

[6] The GNU Project. *GNU Compiler Collection*. Computer software. URL: https://gcc.gnu.org/.

[7] The LLVM Project. *The LLVM Compiler Infrastructure*. Computer software. URL: https://llvm.org/.

[8] K. Fischer and B. Gärtner. 'The Smallest Enclosing Ball of Balls: Combinatorial Structure and Algorithms'. In: *International Journal of Computational Geometry & Applications* 14.04n05 (Oct. 2004), pp. 341–378. ISSN: 0218–1959. DOI: 10.1142/S0218195904001500.

[9] L. F. Greengard and V. Rokhlin. 'A new version of the Fast Multipole Method for the Laplace equation in three dimensions'. In: *Acta Numerica* 6 (Jan. 1997), pp. 229–269. ISSN: 0962–4929. DOI: 10.1017/S0962492900002725.

[10] W. Dehnen. 'A fast multipole method for stellar dynamics'. In: *Computational Astrophysics and Cosmology* 1.1 (2014), pp. 1–23. DOI: 10.1186/s40668-014-0001-7.

[11] L. F. Greengard and V. Rokhlin. 'A fast algorithm for particle simulations'. In: *Journal of Computational Physics* 73.2 (Dec. 1987), pp. 325–348. ISSN: 0021–9991. DOI: 10.1016/0021-9991(87)90140-9.

[12] W. Dehnen. 'A Hierarchical O(N) Force Calculation Algorithm'. In: *Journal of Computational Physics* 179.1 (June 2002), pp. 27–42. ISSN: 0021–9991. DOI: 10.1006/jcph.2002.7026.

[13] R. Yokota. 'An FMM Based on Dual Tree Traversal for Many-Core Architectures'. In: *Journal of Algorithms & Computational Technology* 7.3 (Sept. 2013), pp. 301–324. ISSN: 1748–3018. DOI: 10.1260/1748-3018.7.3.301.

[14] F. G. van Zee and R. A. van de Geijn. ‘BLIS: A Framework for Rapidly Instantiating BLAS Functionality’. In: *ACM Transactions on Mathematical Software* 41.3 (June 2015), pp. 1–33. DOI: 10.1145/2764454.

[15] R. A. van de Geijn, M. Myers and D. Parikh. *LAFF-On Programming for High Performance*. ulaff.net. June 2021. URL: https://www.cs.utexas.edu/users/flame/laff/pfhp/.

[16] The BLIS Project. *BLAS-like Library Instantiation Software Framework*. Computer software. URL: https://github.com/flame/blis.