DASHMM Basic User Guide

The Dynamic Adaptive System for Hierarchical Multipole Moments (DASHMM) is a C++ library providing a general framework for computations using multipole methods. In addition to the flexibility to handle user-specified methods and expansion, DASHMM includes built-in methods and expansions, including the Barnes-Hut (BH) and Fast Multipole Method (FMM), and three expansions implementing the Laplace Kernel used in electrostatics and Newtonian gravitation. Currently, only the Laplace kernel is implemented in DASHMM, in future versions more kernels will be added.

DASHMM is built using the advanced runtime system, HPX-5, but the basic interface to DASHMM does not require any knowledge of how to use HPX-5. Instead, the basic interface handles the distribution of the parallel work. The model of use for the basic interface to DASHMM is to take a serial code, make some calls to the DASHMM library, and in so doing, the program will make use of parallel resources. More options are available in the DASHMM Advanced User Guide, including interoperability with legacy MPI codes, or use in HPX-5 aware applications.

In the following, snippets of code, or the names of code constructs will be set in a fixed width font. For example, main().

Unless otherwise indicated, every construct presented in this guide is a member of the dashmm namespace.

This document covers the basic use of the DASHMM library. More functionality is exposed through the advanced interface to DASHMM, including the ability to define and register methods and expansions with the library. For instructions on the advanced interface, please see the DASHMM Advanced User Guide. Instructions for installing and building applications using the library can be found in the DASHMM Installation Guide. Finally, the latest information, resources and tutorials can be found at the DASHMM webpage: https://www.crest.iu.edu/projects/dashmm/

Types Defined by DASHMM

DASHMM provides a few types that used throughout the system. The following are those needed in the basic interface.

Return codes from DASHMM library calls are all of the ReturnCode (non-scoped) enumeration type. The possible values are kSuccess, kRuntimeError, kIncompatible, kAllocationError, kInitError, kFiniError, and kDomainError. See below for cases in which these values might be returned.

Objects living in the global address space provided by DASHMM have handles of the type ObjectHandle. These can be thought of like an opaque pointer. Routines that create objects and return references to those objects will return ObjectHandles. Routines that require references to objects will take ObjectHandles.

Specification of a particular multipole method is via a class derived from an abstract base class, Method. Typically, unless the user is implementing their own methods, routines in DASHMM will use pointers to the base class. For details on the required interface for a derived Method, and how to register new methods with DASHMM, please see the DASHMM Advanced User Guide.

Specification of a particular multipole expansion is via a class derived from an abstract base class, Expansion. Typically, unless the user is implementing their own expansions, routines in DASHMM will use pointers to the base class. For details on the required interface for a derived Expansion, and how to register new expansions with DASHMM, please see the DASHMM Advanced User Guide.

Basic Interface

The basic interface to DASHMM comprises seven functions. Two for starting and stopping the runtime system, four for providing data to, and getting data from, DASHMM, and evaluate, which performs the multipole method calculation.

ReturnCode init(int \*argc, char \*\*\*argv)

To start the runtime system, a user must call init and provide references to the command line arguments used to launch the program. The runtime has some settings that can be controlled from the command prompt, and so these must be handed to the runtime. For a list of command line arguments that the runtime accepts, please see the DASHMM Advanced User Guide. After init, any runtime related command line argument will have been removed from argv, and argc will be updated accordingly. In addition to setting up the runtime, init will setup some internal bookkeeping for the DASHMM library.

All other calls to DASHMM library routines must occur after the call to init. Only a single call to init is allowed in any program that makes use of DASHMM.

On successful initialization of the runtime and of DASHMM, kSuccess is returned. Otherwise kInitError is returned indicating some problem.

ReturnCode finalize()

Before finishing a program, the user must call finalize to shut down the runtime, and to free the DASHMM specific resources acquired. Once finalize returns, the user’s program is free to continue performing any other work required, but all DASHMM and runtime resources will have been destroyed. So any data that is to be read from DASHMM’s internal state must occur before this call.

All calls to DASHMM library routines must occur before the call to finalize. Only a single call to finalize is allowed in any program that makes use of DASHMM.

If finalize is successful, kSuccess will be returned. Otherwise, kFiniError will be returned.

ReturnCode allocate\_array(size\_t count, size\_t size,

ObjectHandle \*obj)

To provide data to DASHMM, one must make use of array objects. This function will request an array object that can hold count objects, each with a size in bytes of size. If DASHMM is successful in creating an array of the requested size, a handle to that array will be returned via the parameter obj.

This routine provides no user control of the distribution of the records in the array, leaving it instead to DASHMM to provide a good guess. Further, the distribution of the data may change during execution.

This routine returns kSuccess on success, kRuntimeError if there is an error from the runtime, and kAllocationError if the request fails because of a lack of available resources.

ReturnCode deallocate\_array(ObjectHandle obj)

This function instructs DASHMM to reclaim the resources used by an array object. The object handle provided must be a handle to an object allocated with allocate\_array. It is an error to use array\_put or array\_get after the object has been deallocated.

On success, this routine return kSuccess. If there is an error from the runtime, this routine returns kRuntimeError. If the provided object handle is not an array object, or has not been allocated by the user using DASHMM routines, kDomainError is returned.

ReturnCode array\_put(ObjectHandle obj, size\_t first, size\_t last,

void \*in\_data)

Once an array object is created, to copy values into that array so that DASHMM might use them (in evaluate, for example) one must use array\_put. The provided in\_data is copied into the array object with handle obj, in the records in the range [first, last). It is the user’s responsibility to assure that the buffer pointed to by in\_data contains sufficient data to fill the given number of records.

This routine returns kSuccess on successful put, kRuntimeError if there is an error with the runtime, or kDomainError if the provided object handle is not an array, or if there are problems with the given range (e.g. last is beyond the end of the array).

ReturnCode array\_get(ObjectHandle obj, size\_t first, size\_t last,

void \*out\_data)

Once DASHMM has produced results into an array, the data may be obtained from DASHMM using array\_get. Similar to array\_put, the array object is specified by its handle, obj. The data retrieved is from records in the range [first, last). The data is placed into out\_data. It is the user’s responsibility to assure that out\_data has sufficient capacity for the retrieved data.

The routine returns kSuccess on success, kRuntimeError if there is a runtime error, or kDomainError if the provided object handle is not an array, or if the range specified is incompatible with the underlying array.

ReturnCode evaluate(ObjectHandle sources, int spos\_offset,

int q\_offset, ObjectHandle targets,

int tpos\_offset, int phi\_offset,

int refinement\_limit, Method \*method,

Expansion \*expansion)

The central call in the basic interface to DASHMM is evaluate. This routine performs the multipole method evaluation, computing the potential at the specified targets as a result of the specified sources. This will use the provided method, and the provided expansion of the potential in question. The built-in methods can be obtained from some methods covered below.

The source points are provided via a DASHMM array object. The user provides a handle to the array, the offset in each record to the position (three double values stored contiguously), spos\_offset, and the offset in the record to the charge (a double), q\_offset. Similarly, the user must specify an object handle to the DASHMM array containing the target records, an offset in each target record to the target position (three double values stored contiguously), tpos\_offset, and the offset in each target record to the location to save the computed potential value (a double), phi\_offset.

It is important to note that DASHMM will, when beneficial, reorder the sources and targets during the evaluation, so it is recommended that users keep an identifier in the source and target records.

During the evaluation, two hierarchical space-partitioning trees are constructed, one each for the sources and targets. The refinement\_limit specifies the refinement termination criterion. If a given tree node contains fewer than the refinement\_limit, the partitioning halts.

This routine return kSuccess on successful evaluation, kIncompatible if the specified method and expansion cannot be used together, or kRuntimeError if there is some error in the runtime.

Built-in Methods

DASHMM currently provides two built-in methods for immediate use, the Barnes-Hut Method and the Fast Multipole Method. Instances of these methods can be obtained through two members of the dashmm namespace.

Method \*bh\_method(double theta)

The returned Method object will be usable anywhere in DASHMM requiring a method to be specified. This instance of the BH method will use the provided theta as the critical angle for deciding if a given expansion is usable. This criterion matches exactly the criterion introduced by Barnes & Hut[[1]](#footnote-1).

Method \*fmm\_method()

The returned Method object will be usable anywhere in DASHMM requiring a method to be specified. The user will not need to specify the error tolerance with this function. For FMM, the error tolerance is instead specified by the expansion; the more terms in the expansion, the lower the error. See Built-In Expansion, below for details.

Method \*fmm\_exp\_method()

The returned Method object will be usable anywhere in DASHMM requiring a method to be specified. This version of FMM uses the merge-and-shift technique[[2]](#footnote-2). The user will not need to specify the error tolerance with this function. For FMM, the error tolerance is instead specified by the expansion; the more terms in the expansion, the lower the error. See Built-In Expansions, below for details.

Built-in Expansions

DASHMM provides one built-in kernel, the Laplace kernel. This potential represents the potential of a point charge in electrostatics, or a point mass in Newtonian gravitation. This kernel is exposed with three different variations of the multipole expansion.

Expansion \*laplace\_COM\_expansion()

This expansion is an expansion about the center of mass of the represented sources. This expansion includes contributions up to the quadrupole term, but because of the expansion about the center of mass, the dipole term is identically zero. This expansion is well suited to gravitation, and should not be used for problems with both signs of charge. This expansion is not compatible with FMM; this expansion does not implement all of the required operations for use with FMM.

Expansion \*laplace\_sph\_expansion(int n\_digits)

This expansion is intended for use with the standard FMM method (obtained from fmm\_method). This expansion implements all relevant operations, and provides a variable length expansion so that the user provided accuracy limit, given by n\_digits, can be reached.

Expansion \*laplace\_sph\_exp\_expansion(int n\_digits)

This expansion is intended for use with the merge-and-shift FMM (obtained from fmm\_exp\_method). This expansion implements all relevant operations, and provides a variable length expansion so that the user provided accuracy limit, given by n\_digits, can be reached. Currently this expansion supports 3 and 6 digits of accuracy. For accuracy requirements less than this, the expansion will round up to the next available.

1. J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. Nature, 324, 446-449 (1986) [↑](#footnote-ref-1)
2. H. Cheng, L. Greengard, and V. Rokhlin. A Fast Adaptive Multipole Algorithm in Three Dimensions, Journal of Computational Physics, 155, 468-498 (1999) [↑](#footnote-ref-2)