DiMEPACK — A Cache-Aware Multigrid Library
User Manual

Wolfgang Karl, Markus Kowarschik, Ulrich Rüde, Christian Weiβ
DiMEPACK:
A Cache-Aware Multigrid Library
User Manual
Version 1.0

Wolfgang Karl†, Markus Kowarschik‡, Ulrich Rüde†, Christian Weiß†
† Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR–TUM)
Technische Universität München, Germany
{weisss,karlw}@cs.tum.edu
‡ Lehrstuhl für Informatik 10 (Systemsimulation)
Universität Erlangen–Nürnberg, Germany
{kowarschik,rude}@cs.fau.de

March 9, 2001

Abstract
The efficient execution of numerically intensive codes often suffers from high memory access times. There is no doubt about the fact that moving data nowadays is much more expensive than processing data. Thus today’s computer architectures employ hierarchical memory structures with usually several levels of cache memories, which can provide data to the CPU much faster than main memory components. However, efficient execution can only be expected if the code respects the memory design of the underlying architecture. Unfortunately, even modern compilers are not very successful in performing data locality optimizations to enhance the performance of the codes, so that most of this effort is left to the programmer.

DiMEPACK is a C++ library containing cache-optimized multigrid routines for the numerical solution of partial differential equations. DiMEPACK can handle constant-coefficient problems on rectangular domains. It implements a set of highly tuned red-black Gauss–Seidel smoothers as well as cache-aware intergrid transfer operators. In order to reduce the number of cache conflict misses, which especially arise as soon as cache blocking techniques are applied, we have introduced various array padding heuristics. Furthermore, we reduce the total number of arithmetic operations by providing dedicated routines for special cases, like e.g. homogeneous problems.

The DiMEPACK interface is written in C++ whereas the computationally expensive parts of the code are implemented in Fortran 77 for the sake of efficiency.
## Contents

1 Introduction .......................... 5  
   1.1 Motivation ................................ 5  
   1.2 Functionality of DIMEPACK ................. 5  
   1.3 Neumann Boundary Conditions ................. 7  
       1.3.1 Equations for Neumann Boundary Nodes .......................... 7  
       1.3.2 Intergrid Transfer Operators Along Neumann Boundaries ......... 8  
   1.4 Library Overview .......................... 9  

2 Installing and Compiling ................. 10  
   2.1 Make and Build .......................... 10  
   2.2 Compilation Options ......................... 10  
       2.2.1 Debug Options .......................... 11  
       2.2.2 Code Optimization Options ................... 11  
       2.2.3 Miscellaneous Options ..................... 11  
   2.3 Environment Variables ..................... 12  

3 Running DIMEPACK ......................... 12  
   3.1 Data Types .......................... 13  
       3.1.1 Two-dimensional Grid Function .................. 13  
       3.1.2 Boundary Specification ....................... 14  
       3.1.3 Norm and Restriction Types .................. 14  
   3.2 Multigrid Functions ...................... 15  
   3.3 Utility Functions .......................... 17  
   3.4 Example .......................... 17  

4 Known Bugs .......................... 18  
   4.1 Marginal Differences in Floating Point Operation Results ............ 18
1 Introduction

1.1 Motivation

DiMEPACK is a C++ library of cache-optimized multigrid routines for the solution of two-dimensional elliptic partial differential equations (PDEs). So far, DiMEPACK can handle constant-coefficient problems on structured grids. DiMEPACK was developed within the DiME (Data-local iterative methods) research project. The DiME research project is a joint project of the Technische Universität München and the Universität Erlangen–Nürnberg. Our work is presently being funded in part by the Deutsche Forschungsgemeinschaft (DFG), grants Ru 422/7-1,2,3.

In the following we will motivate our research on the design of cache-aware multigrid algorithms. Then, we will describe the functionality of the DiMEPACK library in Section 1.2. In Section 1.3 we will explain how we treat Neumann boundary conditions. We will conclude this chapter with a brief overview of the DiMEPACK library.

The research on cache-aware numerical methods and the development of the DiMEPACK library have been motivated by two independent observations.

- **Computer Architecture:**
  There is no doubt about the fact that the speed of computer processors has been increasing and will even continue to increase much faster than the speed of memory components. As a general consequence, current memory chips based on DRAM technology cannot provide the data to the CPUs as fast as necessary. This memory bottleneck often results in significant idle periods of the processors and thus in very poor code performance compared to the theoretically available peak performances of the machines under consideration. To mitigate this effect modern computer architectures use cache memories which keep data that is frequently used by the CPU. Caches are usually based on SRAM chips which on the one hand are much faster than DRAM components, but on the other hand have comparatively small capacities, for both technical and economical reasons. Most of today's RISC-based workstations even use several levels of caches (one to three levels are common). Efficient execution can therefore be achieved only if the hierarchical structure of the memory subsystem (including main memory, caches and the processor registers) is respected by the code, especially by the order of memory accesses.

- **Numerical Mathematics:**
  It has been demonstrated that — at least from a theoretical point of view — multigrid methods are among the most efficient algorithms for the solution of large systems of linear equations arising e.g., in the context of the numerical solution of PDEs [Bra84, Hac85, Tos01]. Multigrid techniques belong to the class of iterative methods, which means that the underlying data set, which in general is very large, is repeatedly processed several times.

It is our principal goal to investigate in how far multigrid methods can be restructured in order to respect the hierarchical memory design of modern computer architectures and thus significantly speed up their execution.

We have investigated the data access optimization techniques loop fusion as well as one- and two-dimensional loop blocking. Furthermore, we use sophisticated implementations combining smoothing steps and intergrid transfer operations, i.e., residual restriction and error prolongation.

In order to avoid the occurrence of severe cache conflict misses, DiMEPACK also implements array padding heuristics [RT98a, RT98b]. In contrast to loop fusion and loop blocking, array padding is a data layout optimization technique.

Our results and optimization techniques are described in detail in [Rfl97, SR97, SRW97, Rfl98, DHH+00a, WKKR99, WHS00, DRB00, KRW00, DHH+00b, DHH+00]. A more popular description of our work has recently been presented in [DHH+00a, DHH+00b]. Please feel free to contact the authors of this report if you have any further questions. We further recommend that you visit our project home page:

http://wwwbode.in.tum.de/Par/arch/cache

1.2 Functionality of DiMEPACK

In the following we name and briefly explain the relevant features of the DiMEPACK library. Detailed descriptions of the mathematical components are out of the scope of this report and can be found in [BHM00, Tos01], for example.
\begin{itemize}
  \item Multigrid correction scheme:
    \texttt{DiMEPACK} implements a standard multigrid correction scheme, where on each coarser level corrections for the corresponding finer level are computed. FAS multigrid is not supported.
  
  \item 5–point and 9–point stencils:
    \texttt{DiMEPACK} can handle both 5–point stencils and 9–point stencils.
  
  \item Standard grid coarsening:
    The mesh widths \( h_x \) and \( h_y \) of a coarser grid are twice as large as the mesh widths \( h'_x \) and \( h'_y \) of the next finer grid: \( h_x = 2h'_x \), \( h_y = 2h'_y \). Semi-coarsening is not implemented.
  
  \item V–cycles and full multigrid:
    \texttt{DiMEPACK} implements both standard multigrid V–cycles and full multigrid (FMG, nested iteration). These two methods correspond to two different library functions, which will be described in Section 3.2.
  
  \item Constant–coefficient problems:
    So far, \texttt{DiMEPACK} can only handle constant–coefficient problems. The user specifies either the five or the nine entries of a matrix row corresponding to an interior grid node.
  
  \item Rectangular domains:
    \texttt{DiMEPACK} can only handle problems on rectangular domains. Different mesh widths in directions \( x \) and \( y \) are supported. However, the user is responsible to make sure that the standard multigrid components provided by \texttt{DiMEPACK} are applicable to the specified problem.
  
  \item Boundary conditions:
    Each of the four boundary conditions can be of Dirichlet or Neumann type. Again, it is up to the user to provide a reasonable problem specification. See Section 1.3 for further details.
  
  \item Problems involving singular matrices:
    In order to handle problems which yield singular matrices the user must set a flag which guarantees that the solution of the problem on the coarsest grid is fixed in the south–west corner of the rectangular domain, see Section 3.2. Otherwise, the direct solver will fail for the problem on the coarsest grid.

    One important example is Poisson's equation \(-\Delta u = f\) with Neumann boundary conditions along each of the four sides of the rectangular domain. If this differential operator is discretized e.g. using the standard 5–point stencil

    \[
    \frac{1}{h^2} \begin{bmatrix}
      -1 & 4 & -1 \\
      -1 & 4 & -1 \\
      -1 & 4 & -1 \\
    \end{bmatrix}
    \]

    on an equidistant grid with mesh width \( h \), the resulting matrix is singular. This can easily be seen since the sum of the entries of each matrix row is 0, and thus \((1, \ldots, 1)^T\) is an eigenvector corresponding to the eigenvalue 0. This problem can be handled by setting the \texttt{fixSolution} flag to true. See [BP94] for mathematical details on iterative schemes applied to singular systems.
  
  \item Pre– and post–smoothing iterations:
    The numbers of pre–smoothing iterations and post–smoothing iterations must be specified by the user. \texttt{DiMEPACK} provides a Gauss–Seidel/SOR smoother based on a red–black ordering of the unknowns. The use of suitable relaxation parameters can help to obtain good smoothing properties even in the case of moderately anisotropic problems, see Section 3.3 and [Yav96].
  
  \item Number of grid levels:
    The user can specify the number of grid levels. The amount of levels must be equal or greater than 2. However, \texttt{DiMEPACK} is also able to automatically chose the number of grid levels. In this case, the maximum number of grid levels is used.
  
  \item Restriction operators:
    \texttt{DiMEPACK} implements both half–weighting and full–weighting.
\end{itemize}
• **Prolongation operator:**
  The prolongation of the errors from coarser to finer grids is done by linear interpolation.

• **Direct solution of the coarsest problems:**
  The problems on the coarsest grid are solved directly. In the beginning the matrix corresponding to the coarsest problem is split into two triangular factors. This is either done by a LU decomposition or — in the case of a symmetric and positive definite matrix — by a Cholesky decomposition. In the course of the iterative process, the coarse-grid problems are solved by forward-backward solution steps.
  
  Both the decomposition and the forward-backward solution are performed using appropriate routines from the LAPACK library [ABB+99]. It is therefore inevitable that both LAPACK and BLAS are installed on the underlying machine.

• **Stopping criteria:**
  In the case of standard multigrid V-cycles the computation stops
  
  – as soon as the norm of the current residual reaches a given tolerance, or
  
  – as soon as a given number of multigrid iterations has been performed.

  In the FMG case additional V-cycles on the finest level are performed

  – until the residual reaches a given tolerance, or

  – a maximum number of additional V-cycles has been performed.

  The tolerance can be measured in the discrete L2 norm or in the maximum norm. See Section 3.2 for an explanation how the residual norms are specified. The computation of the norm after each iteration, however, is computational intensive. Therefore, DiMEPACK allows this feature to be disabled by the compiler option DIME_COMPUTE_NORM. See Section 2.2.3 for further information.

• **Precision of floating-point arithmetic:**
  DiMEPACK works with either single precision or double precision floating-point numbers. By properly setting the corresponding environment variable DIME_REAL the user determines the type of floating-point representation before compiling the DiMEPACK library, see Section 2.3.

• **Gnuplot interface:**
  DiMEPACK can produce a lot of debugging information, including intermediate solutions and right-hand sides on all grid levels, see Section 2.2.1.

1.3 Neumann Boundary Conditions

This section explains how we treat Neumann boundary conditions in the DiMEPACK library. Some hints how to treat Neumann boundaries can be found in [BHL+92, BHM00]. Note that our treatment of Neumann boundaries is based on a finite difference approach and yields boundary stencils that are different from those obtained for finite element discretization using rectangular elements and bilinear basis functions.

1.3.1 Equations for Neumann Boundary Nodes

We use second-order central differences in order to approximate the external normal derivatives at grid nodes along Neumann boundaries. This yields 4- or 6-point stencils, respectively, for inner (i.e., non-corner) Neumann boundary points, depending on whether a 5-point or 9-point stencil discretization of the differential operator at inner grid points is used. In the following examples, the values $sw, so, se, ve, oe, ea, me, no, we$ denote the constant entries in the bands of the matrix.

**Example:** Neumann condition along the west boundary:

• 5-point discretization:

  $$
  \begin{bmatrix}
  \cdot & no & \cdot \\
  \cdot & oe & ea + we \\
  \cdot & so & \cdot 
  \end{bmatrix}
  $$
• 9-point discretization:

\[
\begin{bmatrix}
  . \ & no \ & ne + nw \\
  . \ & ce \ & ea + we \\
  . \ & so \ & se + sw \\
\end{bmatrix}
\]

The other boundaries of the rectangular domain are treated analogously.

In a corner of the rectangular domain where two Neumann boundaries meet, we obtain 3- or 4-point stencils, respectively. This is due to the fact that in these corner points two external normal derivatives are approximated using central differences.

**Example:** Neumann condition in the south-west corner:

• 5-point discretization:

\[
\begin{bmatrix}
  . \ & no \ & so \\
  . \ & ce \ & ea + we \\
\end{bmatrix}
\]

• 9-point discretization:

\[
\begin{bmatrix}
  . \ & no \ & so \ & ne + nw + se + sw \\
  . \ & ce \ & ea + we \\
  . \ & . \ & . \\
\end{bmatrix}
\]

The other corners of the rectangular domain are treated analogously.

Inhomogeneous Neumann boundaries and homogeneous Neumann boundaries are treated equally, except that the right-hand sides of the corresponding boundary nodes are adapted to the given non-zero derivatives. If possible, equations for Neumann boundary points on the coarsest grid are scaled properly in order to maintain the symmetry of the matrix.

### 1.3.2 Intergrid Transfer Operators Along Neumann Boundaries

In the previous section we have described how we obtain the equations for the grid nodes along the Neumann boundaries of the domain. Now, we explain how the restriction operators \( R_h \) compute the right-hand sides for coarse-grid points on Neumann boundaries and how the corrections for solutions on finer grids are computed by the interpolation operators \( I_h \).

**Restriction.** DiMEPACK implements two different restriction operators: half weighting and full weighting. A more detailed analysis of these operators can e.g. be found in [ST86]. Using the common stencil notation for intergrid operators, these restriction operators for inner grid nodes look as follows:

• Full weighting:

\[
\frac{1}{16} \begin{bmatrix}
  1 & 2 & 1 \\
  2 & 4 & 2 \\
  1 & 2 & 1 \\
\end{bmatrix}
\]

• Half weighting:

\[
\frac{1}{8} \begin{bmatrix}
  . & 1 & . \\
  1 & 4 & 1 \\
  . & 1 & . \\
\end{bmatrix}
\]

According to [BJL+92], we use the following operators for non-corner Neumann boundary points along the west boundary of the domain:

• Full weighting:

\[
\frac{1}{16} \begin{bmatrix}
  . & 2 & 2 \\
  . & 4 & 4 \\
  . & 2 & 2 \\
\end{bmatrix}
\]

• Half weighting:

\[
\frac{1}{8} \begin{bmatrix}
  . & 1 & . \\
  . & 4 & 2 \\
  . & 1 & . \\
\end{bmatrix}
\]
It is obvious that the other three boundaries can be treated analogously.

Corner nodes (where two Neumann boundaries meet) are treated analogously as well. E.g., for the north–west corner we obtain the following restriction operators:

- Full weighting:

\[
\frac{1}{16} \begin{bmatrix}
1 & 1 & 1 \\
1 & 4 & 4 \\
1 & 4 & 4 \\
\end{bmatrix}
\]

- Half weighting:

\[
\frac{1}{8} \begin{bmatrix}
1 & 1 & 1 \\
1 & 4 & 2 \\
1 & 2 & 1 \\
\end{bmatrix}
\]

Note that if both

1. a red–black Gauss–Seidel smoother is used, i.e., the user chooses the relaxation parameter \( \omega = 1 \), and

2. a 5–point stencil is used for discretizing the PDE,

the residuals vanish at the black grid points after every smoothing step, and therefore each of the two restriction operators can be implemented more efficiently by avoiding the computation of the residuals corresponding to black nodes. In particular, this means that the half weighting operator degenerates to the half injection operator

\[
\frac{1}{8} \begin{bmatrix}
1 & 4 & 1 \\
4 & 1 & 4 \\
1 & 4 & 1 \\
\end{bmatrix},
\]

if the user chooses the relaxation parameter \( \omega = 1 \).

Note that this simplification is not correct as soon as \( \omega \neq 1 \) is required or a 9–point operator is used. In each of these cases, the residuals at the black nodes generally do not vanish after a red–black Gauss–Seidel sweep.

Furthermore, we multiply the restricted residuals with an additional factor 4 in order that we can use the fine-grid coefficients for the coarser system as well. This is possible since DiMEPACK merely implements standard coarsening, which means that the mesh widths in both directions \( x \) and \( y \) are doubled when recursively setting up the coarse–grid systems from the corresponding fine–grid systems. Keep in mind that DiMEPACK can only handle problems with constant coefficients.

**Interpolation.** DiMEPACK implements bilinear interpolation operators in order to prolongate the coarse–grid corrections to the finer grids. This is done in a straightforward manner and thus needs no further explanation.

However, it should be noted that the treatment of Neumann boundaries, which we have previously explained, violates the requirement

\[
I_h^H \cdot c \cdot (I_h^H)^T \in \mathbb{R},
\]

which often occurs in the context of multigrid algorithms in order to maintain any potential symmetry of the whole multigrid iteration matrix [ST86].

### 1.4 Library Overview

The DiMEPACK package consists of a main directory and a number of subdirectories:

- padding
- smoother
- restriction
- interpolation
- post–coarse–grid–ops
- pre–coarse–grid–ops
The main directory contains the source code and header files for the C++ user interface. The subdirectory *padding* contains C++ source and header files for an array padding library used by *DiMEPACK* to avoid conflict misses. The other subdirectories contain Fortran 77 source code for standard and optimized smoothing, standard intergrid transfer operators, and optimized intergrid transfer operators. The computationally intensive parts are implemented in Fortran 77 for the sake of efficiency. The highly optimized Fortran 77 codes are generated using the UNIX preprocessing tool `m4`.

After *DiMEPACK* has successfully been built, the library file `libdimepack.a` containing the C++ code for the multigrid scheme and all Fortran subroutines can be found in the main directory of the package. Furthermore, the `libpadding.a` library file for padding heuristics used by *DiMEPACK* can be found in the `padding` subdirectory. Every program using *DiMEPACK* has to link both libraries. Interface definitions for both libraries are provided in the `dimepack.h` include file which is also located in the main directory.

Presently, *DiMEPACK* has been ported to the following platforms:

- LINUX PCs
- Alpha-based Digital/Compaq workstations running Digital UNIX or Compaq Tru64 UNIX

2 Installing and Compiling

In order to build and use the *DiMEPACK* library the following components must be installed:

- GNU `gmake` (see e.g., `http://www.fsf.org`)
- An ANSI C++ compiler and the Standard C++ Template Library (STL)
- A Fortran 77 compiler
- The LAPACK and BLAS library (see e.g., `http://www.netlib.org`). These libraries are necessary because we use LAPACK routines in order to solve the linear systems on the coarsest grids directly.
- The UNIX `m4` preprocessor (see e.g., `http://www.fsf.org`). We need `m4` because the Fortran 77 codes which implement the Gauss-Seidel smoother, the residual restriction, the error prolongation and the interpolation are generated from macros in the course of the compilation process.
- The Korn shell `ksh`. It is only used to simplify the generation of the *DiMEPACK* library using the shell script `build`, see Section 2.1.

2.1 Make and Build

The library is built by a standard `gmake` mechanism. To simplify library building for different architectures we furthermore provide a `ksh` script called `build`.

To start the installation, the user may have to modify the file `make.incl.<os_type>` according to the configuration of the machine under consideration. General build options are changed by modifying `make.incl.general`. These options are explained in more detail in Section 2.2. Finally, the user may wish to set or alter the values of the environment variables `DIME_OPTIMIZED` and `DIME_REAL` (see Section 2.3) before starting compilation.

The compilation of the *DiMEPACK* library is performed by executing the `build` command in the *DiMEPACK* main directory. The `build` command requires the operating system type as a parameter. Currently supported operating system types are `linux` and `trux64`.

Example:

```
% ./build linux
```

2.2 Compilation Options

The compilation of *DiMEPACK* is affected by several compilation options. The following compilation options can be defined or undefined by the user by modifying the file `make.incl.general` which is located in the main directory:
2.2 Compilation Options

2.2.1 Debug Options

- **DIME\_NDEBUG:**
  If this flag is not set, a lot of intermediate results — such as right-hand side vectors, corrections, etc. — are written to appropriately named files. Be careful, since this can result in enormous disk space usage and dramatically reduced execution speed.

- **DIME\_DEBUG\_CACHE\_PROPERTIES:**
  If this flag is set, the cache characteristics which are used by DIMEPACK are printed to stdout.

- **DIME\_DEBUG\_DIRECTSOLVE:**
  If this flag is set, debug messages concerning the direct solution of the coarsest systems are printed to stdout.

- **DIME\_DEBUG\_FMGRHS:**
  If this flag is set, the right-hand sides of each grid level occurring in the course of the setup phase of an FMG method are written to files named f-fmg.1lv<level>.dat.

- **DIME\_DEBUG\_NUMLEVELS:**
  If this flag is set, the total number of grid levels in the multigrid hierarchy is printed to stdout.

- **DIME\_DEBUG\_PADDING:**
  If this flag is set, debug messages concerning the application of the array padding heuristics are printed to stdout.

2.2.2 Code Optimization Options

- **DIME\_USE\_MELTED\_OPS:**
  If this flag is set, highly optimized routines for combined smooth-restrict and interpolate-smooth operations are used. If DIME\_USE\_MELTED\_OPS is set, optimized smoothers are implicitly used, no matter what the value of DIME\_USE\_OPTIMIZED\_SMOOTHER is.

- **DIME\_USE\_OPTIMIZED\_SMOOTHER:**
  If this flag is set, the cache-optimized smoothing routines are used.

- **DIME\_DISABLE\_PADDING:**
  If this flag is set, array padding is disabled. It is highly recommended not to set this flag in order to achieve high runtime efficiency.

2.2.3 Miscellaneous Options

- **DIME\_DUMP\_RESULT:**
  If this flag is set, the result of the computation is written to the file u-exit.dat.

- **DIME\_TIMING\_ENABLED:**
  If this flag is set, the runtimes of the DIMEPACK routines are determined and written to stdout.

- **DIME\_COMPUTE\_NORM:**
  If this flag is set, the discrete L2 norm or the maximum norm of the residual vector is computed before the computation starts and after each cycle. The norm type must be specified as an argument to each of the DIMEPACK interfaces, see Section 3.2.

  If this flag is not set, residual norms will not be computed and the corresponding function arguments will be ignored. Of course, dispensing with the residual computation saves execution time. This is highly recommended if the user knows in advance the number of multigrid iterations to be performed.
2.3 Environment Variables

In addition to the compilation options, the following environment variables affect the generation and execution of the DiMEPACK library. Explanations on how to set or modify shell environment variables can be found in appropriate shell manuals.

- **DIME_OPTIMIZED**:
  
  If this variable is set to true, compiler optimizations are enabled and compiler debugging switches are turned off. It is highly recommended to set DIME_OPTIMIZED to true in order to obtain efficient codes.

- **DIME_REAL**:
  
  If this variable is set to float, DiMEPACK uses single precision floating-point numbers. Otherwise, DiMEPACK uses double precision floating-point arithmetic. Single precision arithmetic is often faster but less of accuracy might be involved with it.

- **DIME_CACHE_SIZE**:
  
  Use this variable to specify the size of the processor cache in bytes. This value is needed to determine appropriate array paddings. If this variable is not set, DiMEPACK issues a warning message and uses a cache size of 8 Kbyte by default, corresponding to the size of the L1 cache of a Digital Alpha 21164 CPU. If the underlying architecture has several levels of caches, it is up to the user to decide which cache level DiMEPACK shall be tailored for. This decision may require various performance tests.

- **DIME_CACHE_LINE_SIZE**:
  
  Use this variable to specify the corresponding cache line size in bytes. Like DIME_CACHE_SIZE it is used by DiMEPACK to determine appropriate array paddings. If the variable is not set, DiMEPACK issues a warning message and assumes a cache line length of 32 bytes by default. This corresponds to the L1 cache line size of the Alpha 21164 processor. Again, if the underlying architecture has several levels of caches, this variable should be chosen according to the cache level which is specified by the environment variable DIME_CACHE_SIZE (see above).

3 Running DiMEPACK

Once you have successfully installed and compiled DiMEPACK you are able to use the C++ functions of the DiMEPACK interface. To use DiMEPACK within your projects you have to include the dimspack.h header file and link libdimspack.a and libpaddings.a. You might as well have to link the LAPACK and BLAS library.

The first step when writing programs using DiMEPACK is to give an appropriate problem specification. This includes the specification of the matrix coefficients, boundary types, boundary values, the right hand side of the equation, the solution vector (which may be initialized), and the types of the residual norm and the restriction operator to be used. DiMEPACK provides several data types to support the programmer in this process. Furthermore, several utility functions are implemented for problem specification and debugging purposes.

After the problem has been specified, one of two multigrid functions can be called. The function dpmvCycleConst implements a standard multigrid V-cycle, whereas the function dpmgCycleConst implements a full multigrid scheme (nested iteration).

Finally, before running the program, the shell environment variables DIME_CACHE_LINE_SIZE and DIME_CACHE_SIZE should be set to appropriate values.

In the following we will introduce the data structures, the multigrid functions, and the utility functions. After that we give a short example demonstrating how to call a DiMEPACK multigrid solver. Whenever the data type DIME_REAL is mentioned it either stands for float or double. The actual type will be determined during the DiMEPACK generation phase according to the contents of the environment variable DIME_REAL (see Section 2.3 for details).
3.1 Data Types

3.1.1 Two-dimensional Grid Function

*DiMEPACK* introduces the object type `dpGrid2d` which basically represents a two-dimensional grid function. The C++ class interface is as follows:

```cpp
class dpGrid2d {
  public:
    dpGrid2d(int xDim, int yDim, DIM_REAL hx, DIM_REAL hy);
    ~dpGrid2d();

    dpGrid2d& operator=(const dpGrid2d& rhs);

    inline int getdimx() const; // return number of grid points in x direction
    inline int getdimy() const; // return number of grid points in y direction
    inline DIM_REAL gethx() const; // return grid spacing in x direction
    inline DIM_REAL gethy() const; // return grid spacing in y direction
    inline int getpad() const; // return padding size
    inline DIM_REAL* getmem(); // return C like data structure
    inline void initzero(); // return init grid to zero
    inline DIM_REAL& setval(int x, int y); // set grid value
    inline DIM_REAL getval(int x, int y) const; // get grid value;

  private:
    ...
};
```

This data type hides array padding techniques. Array padding is determined and introduced whenever the constructor of this class is called. The constructor of `dpGrid2d` takes four parameters:

- `xDim`: the number of grid nodes in direction x, including the boundaries,
- `yDim`: the number of grid nodes in direction y, including the boundaries,
- `hx`: the mesh width in direction x, and
- `hy`: the mesh width in direction y.

The grid dimensions `mnp` and `nyp` have to be chosen such that the required number of grid levels can be allocated, see Section 3.2. Otherwise, *DiMEPACK* issues an error message. The following public methods are important for the *DiMEPACK* user. They are used to read and modify the values stored in the grid.

- **void initzero()**: initialize all values to 0.
  **Example:**
  ```cpp
dpGrid2d u(65, 65, 1.0/64, 1.0/64);
u.initzero();
```

- **DIM_REAL& setval(int x, int y)**: set the value at position `x,y`.
  **Example:**
  ```cpp
const int mnp = 65, nyp = 65;
DIM_REAL hx = 1.0/(DIM_REAL) mnp, hy = 1.0/(DIM_REAL) nyp;
dpGrid2d f(mnp, nyp, hx, hy);
for (int y = 0; y < nyp; y++)
  for (int x = 0; x < mnp; x++)
    f.setval(x, y) = sin(2.0*M_PI*x*hx)*sin(2.0*M_PI*y*hy);
```

and
• `DIME_REAL getval(int x, int y);` return the value at position `x,y`.

Example:

```c
DIME_REAL s=f.getval(42,42);
```

### 3.1.2 Boundary Specification

The type of each side of the rectangular domain is specified by the enumeration type `tBoundary`. Possible values are `DIRICHLET` for a Dirichlet boundary and `NEUMANN` for a Neumann boundary. Each side can either be of Dirichlet or Neumann type. The boundary types are passed to the multigrid solver in an array of `tBoundary` values. The boundary type of each of the four sides can be accessed using the macros `dpNorth`, `dpEAST`, `dpSOUTH`, and `dpWEST`.

Example:

```c
tBoundary bTypes[4];
bType[dpNorth]= DIRICHLET;
bType[dpEast ]= DIRICHLET;
bType[dpSouth]= DIRICHLET;
bType[dpWest ]= NEUMANN;
```

// `bType` is passed to the multigrid function ...

The boundary values for each of the four sides of the rectangular domain are specified separately in an array of type `DIME_REAL`. These values are either interpreted as fixed boundary values in the case of Dirichlet boundaries or as external normal derivatives in the case of Neumann boundaries. The address of each boundary value array is placed into an array of pointers which is then passed to the multigrid solver.

Example:

```c
DIME_REAL *bVals[4];

bVals[dpNORTH]=new DIME_REAL[nxp];
bVals[dpSOUTH]=new DIME_REAL[nxp];
bVals[dpEAST ]=new DIME_REAL[nyp];
bVals[dpWEST ]=new DIME_REAL[nyp];

for(int i=0; i<nxp; i++){
    bVals[dpNORTH][i]=0.0;
    bVals[dpSOUTH][i]=0.0;
}

for(int i=0; i<nyp; i++){
    bVals[dpEAST][i]=0.0;
    bVals[dpWEST][i]=0.0;
}

// bVals is passed to the multigrid function ...

delete[] bVals[dpNORTH];
delete[] bVals[dpSOUTH];
delete[] bVals[dpEAST];
delete[] bVals[dpWEST];
```

### 3.1.3 Norm and Restriction Types

`DiMEPACK` allows to stop iterative solving when the residual falls short of a certain tolerance value if the compiler option `DIME_COMPUTE_NORM` was set during library building (see Section 2.2.3 for more information). For that purpose `DiMEPACK` implements two types of vector norms:
3.2 Multigrid Functions

- **discrete L2 norm:**
  \[ ||r||_{L2} = \sqrt{h_x h_y \sum_i r_i} \]
- **maximum norm:**
  \[ ||r||_{\infty} = \max_i |r_i| \]

The stopping criterion to be used is specified by the enumeration type tNorm. Possible values of that type are L2 for the discrete L2 norm and MAX for the maximum norm.

DiMEPACK implements half-weighting and full-weighting. The restriction operator to be used in your program must be specified by the enumeration type tRestrict. Possible values of that type are HW for half-weighting and FW for full-weighting.

3.2 Multigrid Functions

DiMEPACK provides two interface functions which can be called from outside. The prototypes of these functions and the definitions of the new data types can be found in the header file dimepack.h, which is also located in the main directory. Depending on whether the user wants standard multigrid V-cycles or full multigrid cycles to be performed, the following functions have to be invoked:

- **Standard multigrid V-cycles:**
  ```c
  void dpVcycleConst(int nLevels, tNorm nType, DIME_REAL epsilon, int maxIt,
  dpGrid2d *u, bool isInitialized, dpGrid2d *fIn, int nul,
  int n2, int nCoeff, DIME_REAL *matrixCoeff, tBoundary *bTypes,
  DIME_REAL **bWals, tRestrict rType, DIME_REAL omega,
  const bool fixSolution=false)
  ```

- **Full multigrid (FMG, nested iteration):**
  ```c
  void dpFMGcycleConst(int nLevels, tNorm nType, DIME_REAL epsilon,
  int maxAddIt, dpGrid2d *u, dpGrid2d *fIn, int nul,
  int n2, int gamma, int nCoeff, DIME_REAL *matrixCoeff,
  tBoundary *bTypes, DIME_REAL **bWals, tRestrict rType,
  DIME_REAL omega, const bool fixSolution=false)
  ```

We will explain the meaning of the parameters of the two functions in the following:

- **nLevels:** (input)
  The total number of grid levels to be used. If this value is too large, DiMEPACK automatically uses the maximum number of levels. If this value is 0, the maximum number of grid levels is used, too. There have to be at least two grid levels.

- **nType:** (input)
  Specifies the type of the norm to be used for the stopping criterion. Legal values are L2 (discrete L2 norm) and MAX (maximum norm), see Section 3.1.3. For this purpose, DiMEPACK provides the enumeration data type tNorm.

- **epsilon:** (input)
  Specifies the tolerance for the algebraic error. If the compiler macro DIME_COMPUTE_NORM is enabled, the multigrid iteration will be performed until the norm of the residual becomes less than the value of this parameter. If DIME_COMPUTE_NORM is disabled, the value of this parameter is ignored, see again Section 2.2.3.

- **maxIt:** (input)
  The maximum number of multigrid V-cycles to be performed (only for standard multigrid V-cycles).

- **u:** (input/output)
  Pointer to the grid function object, where the solution shall be written to. In the case of the V-cycle scheme this grid function object may provide an initial guess. Set the isInitialized parameter to true to indicate this.
• isInitialized: (input)
  Indicates if the grid function object where the solution shall be stored is already initialized. If the value of this parameter is false, the initial guess will be the constant 0.

• fIn: (input)
  Pointer to the grid function object storing the right-hand side of the equation. Pass the NULL pointer to indicate that the algebraic problem is homogeneous. This will save floating-point operations.

• m1: (input)
  The number of pre-smoothing red-black Gauss-Seidel iterations to be performed. The number must be equal or greater than 1.

• m2: (input)
  The number of post-smoothing red-black Gauss-Seidel iterations to be performed. The number must be equal or greater than 0.

• nCoeff: (input)
  The number of coefficients in the stencil (for an inner grid point). Legal values for this parameter are 5 and 9.

• matrixCoeff: (input)
  The entries of a matrix row corresponding to an inner grid point, i.e., to a grid point with the maximum number of nCoeff of unknown neighboring values. The order of the coefficients is south, west, center, east, north in the case of a 5-point stencil and south–west, south, south–east, west, center, east, north–west, north, north–east for a 9-point stencil.

• bTypes: (input)
  Specifies the types of the four boundaries of the rectangular domain. The order is north boundary, east boundary, south boundary, west boundary. Supported boundary types are DIRICHLET and NEUMANN. For this purpose, DiMEPACK provides the enumeration data type tBoundary. Boundary types may be mixed from side to side.

• bVals: (input)
  Specifies the boundary values for each of the four boundaries of the rectangular domain. These values are either interpreted as fixed boundary values in the case of Dirichlet boundaries or as external normal derivatives in the case of Neumann boundaries.

• rType: (input)
  Denotes the type of operator used to restrict the residuals from a finer grid level to the next coarser grid level. Supported values are HW (half weighting) and FW (full weighting). For this purpose, DiMEPACK provides the enumeration data type tRestrict.

• omega: (input)
  Specifies the relaxation parameter \( \omega \) for the red-black SOR smoother. Good smoothing properties in the case of moderately anisotropic differential operators can be achieved by choosing suitable relaxation parameters [Yav96]. DiMEPACK provides the utility function dCalcOmega to determine a good relaxation parameter \( \omega \), see Section 3.3. If \( \omega = 1.0 \) is specified, a standard red-black Gauss–Seidel smoother is invoked.

• fixSolution: (input)
  Specifies if the solution on the coarsest grid is to be fixed. There are cases where the user wants to solve singular problems, e.g., Poisson's equation with four Neumann boundaries. In order to obtain a regular system on the coarsest grid, this flag must be used. Otherwise, LAPACK will not be able to factorize the corresponding matrix. If this flag is set to true, the solution in the south–west corner of the coarsest grid will be set to 0. Note that this is an arbitrary choice.

• gamma: (input)
  Specifies the cycling parameter in the case of FMG. I.e., the number of V–cycles to be performed after interpolating the approximation of the solution to a finer level of the grid hierarchy.
3.3 Utility Functions

DiMEPACK provides several utility functions, two of which may be of interest to the user. Thus, they will be explained in the following.

- \texttt{DiME\_REAL dpCalcOmega(DIME\_REAL hx, DIME\_REAL hy)}

  This function computes a suitable relaxation parameter \( \omega \) for the Laplacian operator on a moderately anisotropic grid. This is equivalent to a moderately anisotropic behavior of the differential operator itself. The parameters \( hx \) and \( hy \) denote the mesh widths in directions \( x \) and \( y \), respectively. The return value of this function is the recommended relaxation parameter \( \omega \). The calculation of suitable relaxation parameters is based on [Yav96].

- \texttt{void dpPrintGrid(ostream& os, dpGrid2d& grid)}

  This function can be used to write grid data to an \texttt{ostream} object. The second parameter \texttt{grid} specifies the grid object to be printed.

3.4 Example

In this section we give an example showing how the \textit{DiMEPACK} function \texttt{dpVcycleCons} is called. We solve Poisson’s equation on the unit square using a moderately anisotropic grid with \( hx = hy/2 \). We apply the utility function \texttt{dpCalcOmega} to determine a suitable relaxation parameter \( \omega \) for the red-black SOR smoother. We further assume homogeneous Dirichlet boundary conditions. The macro \texttt{DIME\_REAL} either stands for \texttt{float} or \texttt{double}. This depends on whether the \textit{DiMEPACK} library has been built in order to handle single precision or double precision numbers.

// Include the header file containing the function prototypes // and the data type definitions:
#include "dimpack.h"

void runDiMEPACK(void)
{
  // Initialize parameters:
  const int npx=513;  // Number of grid nodes in direction x
  const int npy=257;  // Number of grid nodes in direction y
  const int nLevels=0;  // Use as many levels as possible
  const tNorm nType=L2;  // Use L2 norm
  const DIME\_REAL epsilon=1e-16;  // Required accuracy
  const int maxIt=5;  // Max. number of multigrid V cycles
  const int mul=2;  // Number of pre-smoothing iterations
  const int mu2=2;  // Number of post-smoothing iterations
  const tRestrict rType=FW;  // Full weighting
  const bool fixSol=false;  // Don’t fix the solution on the coarsest grid
  DIME\_REAL omega=dpCalcOmega(hx,hy);  // Get suitable relaxation parameter
  const int nCoeff=5;  // Number of matrix coefficients per row

  // Mesh width in direction x:
  const DIME\_REAL hx=1.0/(DIME\_REAL (npx-1));
  // Mesh width in direction y:
  const DIME\_REAL hy=1.0/(DIME\_REAL (npy-1));

  // Create the solution object and initialize it:
  dpGrid2d u(npx,npy,hx,hy);
  const bool isInitialized=false;

  // Create right-hand side object and initialize it:
  dpGrid2d f(npx,npy,hx,hy);
  for (int y=0; y<npy; y++)
  for (int x=0; x<npx; x++)
    f.setval(x,y)= sin(M_PI*x*hx)*sin(M_PI*y*hy);
}
// Allocate memory for the matrix entries and define them:
DIME_REAL matrixCoeff[nCoeff];
matrixCoeff[0]=1.0/(hx*hy);
matrixCoeff[1]=1.0/(hx*hx);
matrixCoeff[2]=2.0/(hx*hx)+2.0/(hy*hy);
matrixCoeff[3]=1.0/(hx*hx);
matrixCoeff[4]=1.0/(hy*hy);

// Specify boundary types and boundary values:
tBoundary bTypes[4];
for(i=0; i<4; i++)
    bTypes[i]=DIRICHLET;
DIME_REAL * bVals[4];
bVals[dpNORTH]=new DIME_REAL[nxp];
bVals[dpSOUTH]=new DIME_REAL[nxp];
bVals[dpEAST]=new DIME_REAL[nyp];
bVals[dpWEST]=new DIME_REAL[nyp];
for(i=0; i<nxp; i++) {
    bVals[dpNORTH][i]=0.0;
}
for(i=0; i<nyp; i++) {
    bVals[dpEAST][i]=0.0;
    bVals[dpWEST][i]=0.0;
}

// Call the DiMEPACK library function:
dpVcycleCons(nLevels,nType,epsilon,maxIt,&u),isInitialized,&f,m1,m2,nCoeff,
        matrixCoeff,bTypes,bVals,rType,omega,fixSol);

// Clean up:
delete[] bVals[dpNORTH];
delete[] bVals[dpEAST];
delete[] bVals[dpSOUTH];
delete[] bVals[dpWEST];

// Process results:
// ...;
return;
}

See the included files testdp.C and benchdp.C for further examples involving non-homogeneous problems, Neumann boundary conditions, different differential operators, etc.

4 Known Bugs

4.1 Marginal Differences in Floating Point Operation Results

Instruction reordering causes slightly different results when cache-optimized intergrid transfer operations are used. This is due to the fact that addition and multiplication, in general, are not associative operations in finite-precision arithmetic.

As soon as the user defines the compiler macro DIME_USE_MELTED_OPS (see Section 2.2) the results of the computation may not be bitwise identical to the results obtained without the use of this flag although no data dependencies are violated by the optimization techniques. The reason for this is that defining the flag DIME_USE_MELTED_OPS implies different execution orders for the arithmetic operations.

Similar effects occur as soon as the environment variable DIME_OPTIMIZED is set to true. According to
Section 2.3 this means that full compiler optimization is enabled, which again implies different execution orders for the arithmetic operations.

This is not really a bug, but the consequence of the fact that basic algebraic laws do not hold for machine computations.

References


