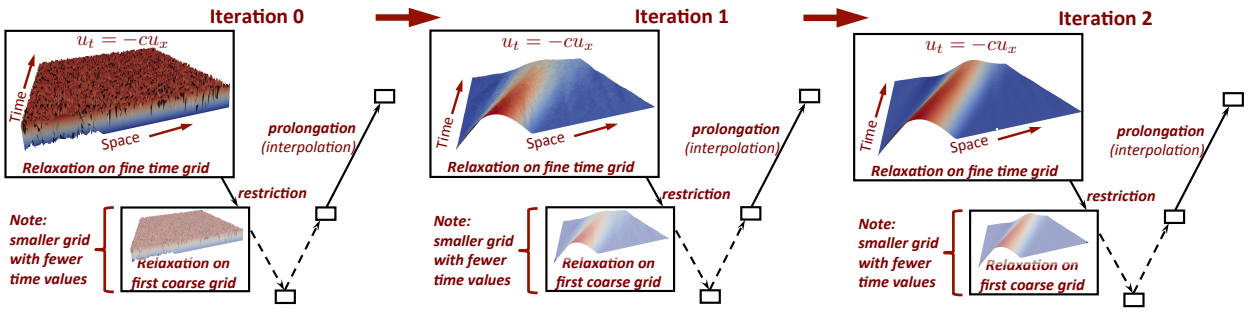
Warp 1.0 Users' Manual



V. A. Dobrev, R. D. Falgout, Tz. V. Kolev,
N. A. Petersson, J. B. Schroder, U. M. Yang,

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
P.O. Box 808, L-561
Livermore, CA 94551

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-******

Contents

1 Abstract

This package implements an optimal-scaling multigrid solver for the linear systems that arise from the discretization of problems with evolutionary behavior. Typically, solution algorithms for evolution equations are based on a time-marching approach, solving sequentially for one time step after the other. Parallelism in these traditional time-integration techniques is limited to spatial parallelism. However, current trends in computer architectures are leading towards systems with more, but not faster, processors. Therefore, faster compute speeds must come from greater parallelism. One approach to achieve parallelism in time is with multigrid, but extending classical multigrid methods for elliptic operators to this setting is a significant achievement. In this software, we implement a non-intrusive, optimal-scaling time-parallel method based on multigrid reduction techniques. The examples in the package demonstrate optimality of our multigrid-reduction-in-time algorithm (MGRIT) for solving a variety of equations in two and three spatial dimensions. These examples can also be used to show that MGRIT can achieve significant speedup in comparison to sequential time marching on modern architectures.

It is **strongly recommended** that you read `Parallel Time Integration with Multigrid` before proceeding.

2 Overview of Warp Algorithm

The goal of Warp is to solve a problem faster than is possible with a traditional time marching algorithm. Instead of sequential time marching, Warp solves the problem iteratively by simultaneously updating the current solution guess over all time values. The initial solution guess can be anything, even a random function over space-time. The iterative updates to the solution guess are done by constructing a hierarchy of temporal grids, where the finest grid contains all of the time values for the simulation. Each subsequent grid is a coarser grid with fewer time values. The coarsest grid has a trivial number of time steps and can be quickly solved exactly. The overall effect is that solutions to the time marching problem on the coarser (i.e., cheaper) grids can be used to correct the original finest grid solution.

To understand how Warp differs from traditional time marching, consider the simple linear advection equation, $u_t = -cu_x$. The next figure depicts how one would typically evolve a solution here with sequential time stepping. The solution propagates sequentially across space as time increases.

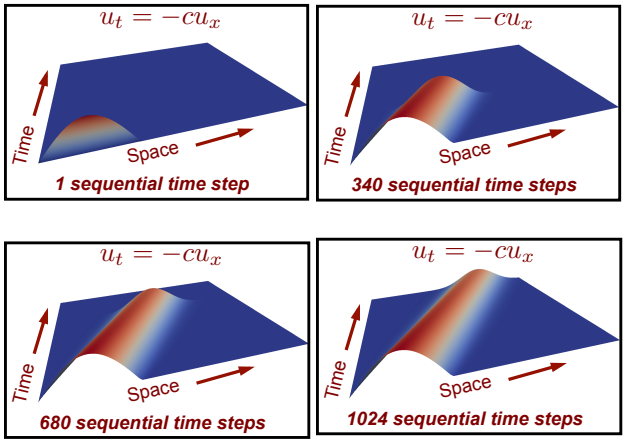


Figure 1: Sequential time stepping.

Warp instead begins with a solution guess over all of space-time, which we let here be random for the sake of argument. A Warp iterations then does

1. Relaxation on the fine grid, i.e., the grid that contains all of the desired time values
    - Relaxation is just a local application of the time stepping scheme, e.g., backward Euler
2. Restriction to the first coarse grid, i.e., a grid that contains fewer time values, say every second or every third time value
3. Relaxation on the first coarse grid
4. Restriction to the second coarse grid and so on...
5. When a coarse grid of trivial size (say 2 time steps) is reached, it is solved exactly.
6. The solution is then interpolated from the coarsest grid to the finest grid

After the cycle is complete, it repeats until the solution is accurate enough. This is depicted in the next figure, where only a few iterations are required for this simple problem.

There are a few important points to make.

- The coarse time grids allow for global propagation of information across space-time with only one Warp iteration (c.f. how the solution is updated from iteration 0 to iteration 1).
- Using coarser (cheaper) grids to correct the fine grid is analagous to spatial multigrid.

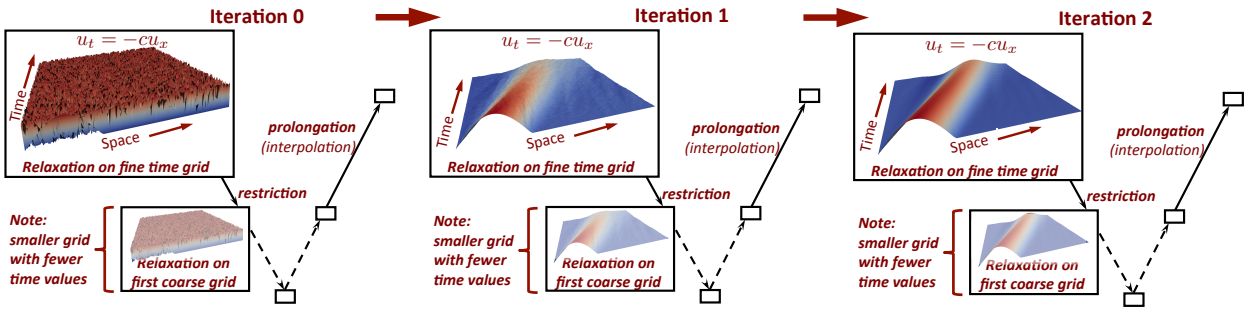


Figure 2: Warp iterations.

- Only a few Warp iterations are required to find the solution over 1024 time steps. Therefore if enough processors are available to parallelize Warp, we can see a speedup over traditional time stepping (more on this later).
- This is a simple example, and Warp is structured to handle variable time step sizes and adaptive time step sizes, and these features will be coming.

To firm up our understanding, let's do a little math. Assume that you have a general ODE,

$$u'(t) = f(t,u(t)), \quad u(0) = u_0, \quad t \in [0,T],$$

which you descretize with the one-step integration

$$u_i = \Phi_i(u_{i-1}) + g_i, \quad i = 1,2,...,N.$$

Traditional time marchine would first solve for $i = 1$, then solve for $i = 2$, and so on. This process is equivalent to a forward solve of this system,

$$A\mathbf{u} \equiv \begin{pmatrix} I & & & \\ -\Phi_1 & I & & \\ & \ddots & \ddots & \\ & & -\Phi_N & I \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_N \end{pmatrix} \equiv \mathbf{g}$$

or

$$A\mathbf{u} = \mathbf{g}.$$

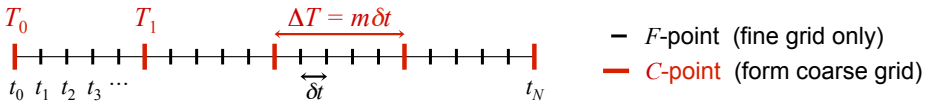
This process is optimal and O(N), but it is sequential. Warp instead solves the system iteratively, with a multigrid reduction method [1] applied in only the time dimension. This approach is

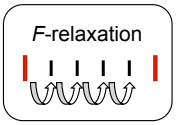
- nonintrusive, in that it coarsens only in time and asks the user to define their own Φ
- optimal and O(N) with a higher constant than time stepping
- highly parallel

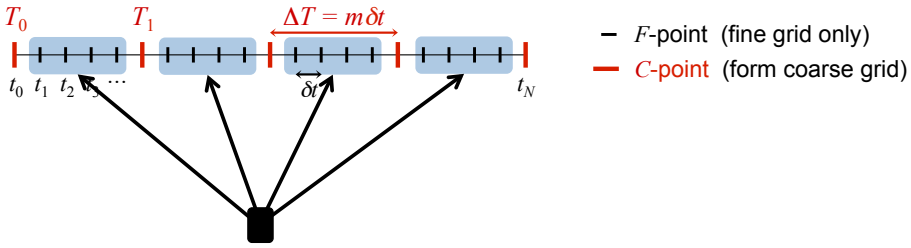
Warp solves this system iteratively by constructing a hierarchy of time grids. We describe the two-grid process, with the multigrid process being a recursive application of the process. We also assume that Φ is constant for notational simplicity.

Warp functions as follows. The next Figure depicts a sample timeline of time values, where the time values have been split into C and F points. C points exist on both the fine and coarse time grid, but F points exist only on the fine time scale. Following the above steps outlining a Warp cycle, the first task is relaxation. The simplest relaxation here alternates between C and F sweeps. An F sweep simply updates time values by integrating with Φ over all the F points from one C point to the next, as depicted here

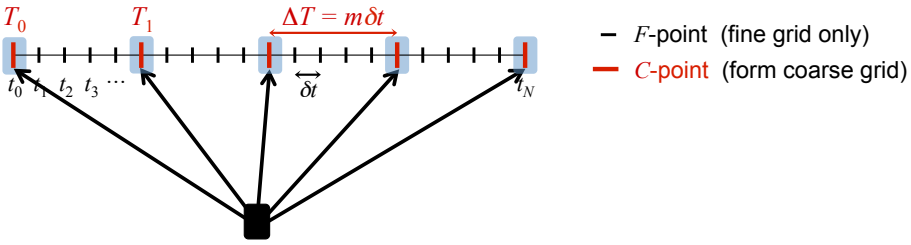
But, such an update can be done simultaneously over all F intervals in parallel, as depicted here







- Update all F-points using time propagator Φ



- Update all C-points using time propagator Φ

Following an F sweep we can also do C sweep, as depicted here FCF or F relaxation will refer to how the relaxation sweeps are carried out. So, we can say

- FCF or F relaxtion is highly parallel.
- But, a sequential component exists equaling the the number of F points between two C points.
- Warp uses regular coarsening factors, i.e., the spacing of C points happens every k points.

After relaxation, then comes coarse grid correction. The restriction operator R maps to the coarse grid by simply injecting values at C points from the fine grid to the coarse grid,

$$R = \begin{pmatrix} I & & \\ 0 & & \\ \vdots & & \\ 0 & & \\ & I & \\ & 0 & \\ & \vdots & \\ & 0 & \\ & & \ddots \end{pmatrix},$$

where the spacing between each I is $m-1$ block rows. This Warp implements an FAS (Full Approximation Scheme)

[1] Ries, Manfred, Ulrich Trottenberg, and Gerd Winter. "A note on MGR methods." Linear Algebra and its Applications 49 (1983): 1-26.

multigrid cycle, and hence the system matrix , solution guess and right-hand-side (i.e., $A, \mathbf{u}, \mathbf{g}$) are all restricted. This is in contrast to linear multigrid which typically restricts the residual equation to the coares grid. We choose FAS because it is `nonlinear` multigrid and allows us to solve nonlinear problems.

The main question here is how to form the coarse grid matrix, which in turn is defined by the coarse grid time stepper Φ_Δ. It is typical to let Φ_Δ simply be Φ but with the coarse time step size Δt. Thus if

$$A = \begin{pmatrix} I & & & \\ -\Phi & I & & \\ & \ddots & \ddots & \\ & & -\Phi & I \end{pmatrix}$$

then

$$A_\Delta = \begin{pmatrix} I & & & \\ -\Phi_\Delta & I & & \\ & \ddots & \ddots & \\ & & -\Phi_\Delta & I \end{pmatrix},$$

where A_Δ has fewer rows and columns than A, e.g., if we are coarsening in time by 2, it has one half as many rows and columns. This coarse grid equation

$$A_\Delta \mathbf{u}_\Delta = \mathbf{g}_\Delta \equiv R\mathbf{g}$$

is then solved. Finally, $\mathbf{u}_\Delta$ is interpolated with P_Φ back to the fine grid, which is equivalent to injecting the coarse grid to the C points on the fine grid, follwed by an F relaxation sweep. That is,

$$P_\Phi = \begin{pmatrix} I & & \\ \Phi & & \\ \Phi^2 & & \\ \vdots & & \\ \Phi^{m-1} & & \\ & I & \\ & \Phi & \\ & \Phi^2 & \\ & \vdots & \\ & \Phi^{m-1} & \\ & & \ddots \end{pmatrix},$$

where m again is the coarsening factor.

This two-grid process is captured with this algorithm. Using a recursive coarse grid solves makes the process multilevel, and halting is done based on a residual tolerance. If the operator is linear, this FAS cycle is equivalent to standard linear multigrid. Note that we represent A as function below, whereas above the notation was simplified for the linear case.

1. Relax on $A(\mathbf{u}) = \mathbf{g}$ using FCF-relaxation
2. Restrict the fine grid approximation and its residual: $\mathbf{u}_\Delta \leftarrow R\mathbf{u}, \quad \mathbf{r}_\Delta \leftarrow R(\mathbf{g} - A(\mathbf{u})$
3. Solve $A_\Delta(\mathbf{v}_\Delta) = A_\Delta(\mathbf{u}_\Delta) + \mathbf{r}_\Delta$
4. Compute the coarse grid error approximation: $\mathbf{e}_\Delta = \mathbf{v}_\Delta - \mathbf{u}_\Delta$
5. Correct: $\mathbf{u} \leftarrow \mathbf{u} + P\mathbf{e}_\Delta$

In summary, a few points are

- Warp is an iterative solver for the global space-time problem.
- The user defines the time stepping routine Φ.

- Warp convergence will depend heavily on how well Φ_Δ approximates Φ^m, that is how well a time step size of $m\delta t = \Delta T$ will approximate m applications of the same time integrator for a time step size of δt. This is a subject of research, but this approximation need not capture fine scale behavior, which is captured by relaxation on the fine grid.
- The coarsest grid is solved exactly, i.e., sequentially, which is a bottleneck. This is an issue for two-level methods like Parareal, [2] but not for a multilevel scheme like Warp where the coarsest grid is of trivial size.
- By forming the coarse grid to have the same sparsity structure and time stepper as the fine grid, the algorithm can recur easily and efficiently.
- Interpolation and restriction are ideal or exact, in that an application of interpolation leaves a zero residual at all F points.

Overview of Warp Code

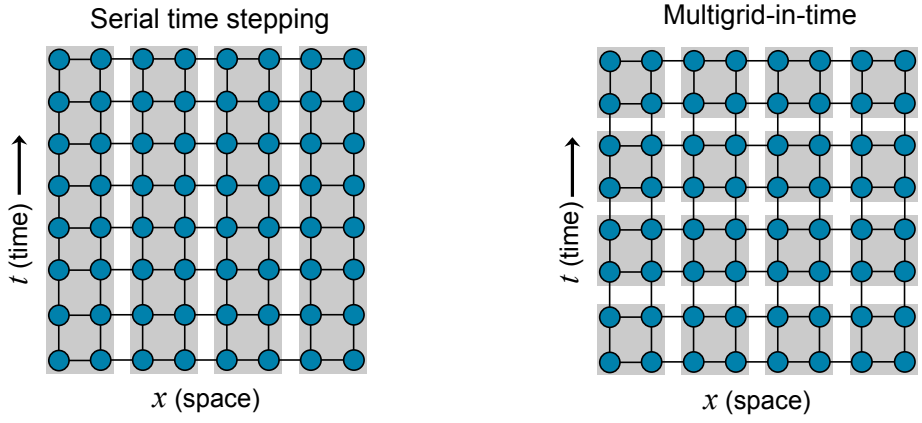
Warp is designed to run in conjunction with an existing application code that can be wrapped per our interface. This application code will implement some time marching type simulation like fluid flow. Essentially, the user has to take their application code and extract a stand-alone time-stepping function Φ that can evolve a solution from one time value to another, regardless of time step size. After this is done, the Warp code takes care of the parallelism in the time dimension.

Warp

- is written in C/C++ and can easily interface with Fortran
- uses MPI for parallelism
- self documents through comments in the source code and through $*$.md files

Parallel decomposition and memory

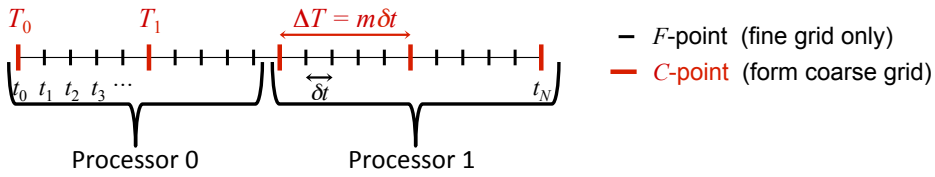
- Warp decomposes the problem in parallel as depicted in this figure. As you can see, traditional time stepping



only stores one time step as a time, but only enjoys a spatial data decomposition and spatial parallelism. On the other hand, Warp stores multiple time steps simultaneously and each processor holds a space-time chunk.

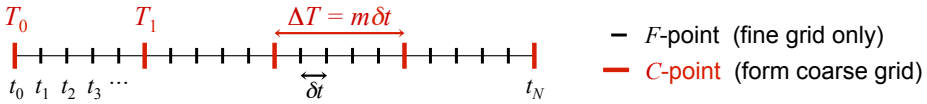
- Warp only handles temporal parallelism and is agnostic to the spatial decomposition. See the warp_Split-Commworld. Each processor owns a certain number of CF intervals of points, as depicted here One interval

[2] Lions, J., Yvon Maday, and Gabriel Turinici. "A"parareal"in time discretization of PDE's." Comptes Rendus de l'Academie des Sciences Series I Mathematics 332.7 (2001): 661-668.



is a C point and all following F points until the next C point. These intervals are distributed evenly on the finest grid.

- Storage is greatly minimized by only stored C points. Whenever an F point is needed, it is generated by F relaxation. That is, we only store the red C point time values. Coarsening can by aggressive with $m = 8, 16, 32$,

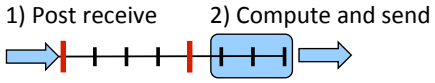


so the storage requirements of Warp are significantly reduced when compared to storing all of the time values.

- In practice, storing only one space-time slab is advisable. That is, solve for as many time steps (say k time steps) as you have available memory for. Then move on to the next k time steps.

Overlapping communication and computation

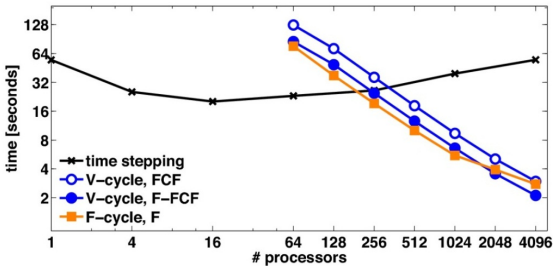
Warp effectively overlaps communication and computation. The main computation kernel of Warp is an F relaxation sweep, and each processor first posts a send at it's left-most C point, and then carries out F relaxation on its right-most interval. If each processor has multiple intervals at this Warp level, this should allow for complete overlap.



Properties of Warp

- Warp applies multigrid to the time dimension
  - Exposes concurrency in the time dimension
  - Potential for speedup is large, 10x, 100x, ...
- Non-intrusive approach, with unchanged time discretization defined by user
- Parallel time integration is only useful beyond some scale. This is evidenced by the experimental results below. For smaller numbers of cores sequential time stepping is faster, but at larger core counts Warp is much faster.
- The more time steps that you can parallelize over, the better your speedup.
- Warp is optimal for a variety of parabolic problems (see the examples).

Here is some experimental data for the 2D heat equation generated by the drive-02 example. Here, the speedup is 10x on a Linux cluster using 4 cores per node, a Sandybridge Intel chipset, and a fast Infiniband interconnect. The problem size was $129^2 \times 16,192$. The coarsening factor $m = 16$ on the finest level and 2 on coarser levels. And various relaxation and V and F cycling strategies are experimented with.



3 A Simple Example

User Defined Structures and Wrappers

As mentioned, the user must wrap their existing time stepping routine per the Warp interface. To do this, the user must define two data structures and some wrapper routines. To make the idea more concrete we give these function definitions from drive-01, which implements a scalar ODE, $u_t = \lambda u$.

The two data structures are:

1. **App**: This holds a wide variety of information and is `global` in that it is passed to every function. This structure holds everything that the user will need to carry out a simulation. Here, this is just the global MPI communicator and few values describing the temporal domain.
```
typedef struct _warp_App_struct
{
   MPI_Comm  comm;
   double    tstart;
   double    tstop;
   int       ntime;

} my_App;
```
2. **Vector**: this defines something like a state vector at a certain time value. It could contain any other information related to this vector needed to evolve the vector to the next time value, like mesh information.
```
typedef struct _warp_Vector_struct
{
   double value;

} my_Vector;
```

And, the user must define a few wrapper routines. The app structure is the first argument to every function.

1. **Phi**: this is the time stepping routine. The user must advance the vector u from time tstart to time tstop. Here advancing the solution just involves the scalar λ. The rfactor_ptr and accuracy inputs are advanced topics.

Importantly, the g_i function (from the Introduction) must be incorporated into Phi, so that $Phi(u_i) \rightarrow u_{i+1})$

```
int
my_Phi(warp_App     app,
       double       tstart,
       double       tstop,
       double       accuracy,
       warp_Vector  u,
       int         *rfactor_ptr)
{
   /* On the finest grid, each value is half the previous value */
   (u->value) = pow(0.5, tstop-tstart)*(u->value);

   /* Zero rhs for now */
   (u->value) += 0.0;
```

```
   /* no refinement */
   *rfactor_ptr = 1;

   return 0;
}
```

1. **Init**: initializes a vector at time t. Here that is just allocating and setting a scalar on the heap.
```
int
my_Init(warp_App     app,
        double       t,
        warp_Vector *u_ptr)
{
   my_Vector *u;

   u = (my_Vector *) malloc(sizeof(my_Vector));
   if (t == 0.0)
   {
      /* Initial guess */
      (u->value) = 1.0;
   }
   else
   {
      /* Random between 0 and 1 */
      (u->value) = ((double)rand()) / RAND_MAX;
   }
   *u_ptr = u;

   return 0;
}
```
2. **Clone**: clones a vector into a new vector
```
int
my_Clone(warp_App     app,
         warp_Vector  u,
         warp_Vector *v_ptr)
{
   my_Vector *v;

   v = (my_Vector *) malloc(sizeof(my_Vector));
   (v->value) = (u->value);
   *v_ptr = v;

   return 0;
}
```
3. **Free**: frees a vector
```
int
my_Free(warp_App    app,
        warp_Vector u)
{
   free(u);

   return 0;
}
```
4. **Sum**: sums two vectors (AXPY operation)
```
int
my_Sum(warp_App    app,
       double      alpha,
       warp_Vector x,
       double      beta,
       warp_Vector y)
{
   (y->value) = alpha*(x->value) + beta*(y->value);

   return 0;
}
```
5. **Dot**: takes the dot product of two vectors
```
int
```

```
my_Dot(warp_App     app,
       warp_Vector  u,
       warp_Vector  v,
       double      *dot_ptr)
{
   double dot;

   dot = (u->value)*(v->value);
   *dot_ptr = dot;

   return 0;
}
```

6. **Write**: writes a vector to screen, file, etc... The user defines what is appropriate output. Notice how you are told the time value of the vector u and even more information in status. This lets you tailor the output to only outputting certain time values and even from various Warp levels and Warp iterations if status is querried.

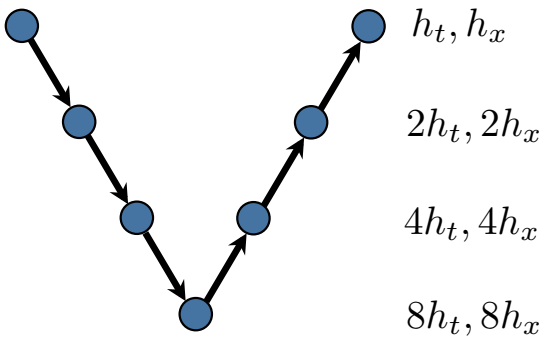
```
int
my_Write(warp_App     app,
         double       t,
         warp_Status  status,
         warp_Vector  u)
{
   MPI_Comm   comm   = (app->comm);
   double     tstart = (app->tstart);
   double     tstop  = (app->tstop);
   int        ntime  = (app->ntime);
   int        index, myid;
   char       filename[255];
   FILE      *file;

   index = ((t-tstart) / ((tstop-tstart)/ntime) + 0.1);

   MPI_Comm_rank(comm, &myid);

   sprintf(filename, "%s.%07d.%05d", "drive-01.out", index, myid);
   file = fopen(filename, "w");
   fprintf(file, "%.14e\n", (u->value));
   fflush(file);
   fclose(file);

   return 0;
}
```

7. **BufSize**, **BufPack**, **BufUnpack**: packs a vector into a `void *` buffer for MPI and then unpacks it from `void *` to vector. Here doing that for a scalar is trivial.

```
int
my_BufSize(warp_App  app,
           int      *size_ptr)
{
   *size_ptr = sizeof(double);
   return 0;
}

int
my_BufPack(warp_App     app,
           warp_Vector  u,
           void        *buffer)
{
   double *dbuffer = buffer;

   dbuffer[0] = (u->value);

   return 0;
}

int
my_BufUnpack(warp_App     app,
             void        *buffer,
             warp_Vector *u_ptr)
```

```
{
   double    *dbuffer = buffer;
   my_Vector *u;

   u = (my_Vector *) malloc(sizeof(my_Vector));
   (u->value) = dbuffer[0];
   *u_ptr = u;

   return 0;
}
```

8. **Coarsen**, **Restrict** (optional): advanced option that allows for coarsening in space while you coarsen in time. This is useful for maintaining stable explicit schemes on coarse time scales and is not needed here. See for instance drive-04 and drive-05 which use these routines.
   These functions allow you vary the spatial mesh size on Warp levels as depicted here



9. Adaptive, variable time stepping is in the works to be implemented. The rfactor parameter in Phi will allow this.

Running Warp

A typical flow in the `main` function to use Warp is to first initialize the app structure

```
/* set up app structure */
app = (my_App *) malloc(sizeof(my_App));
(app->comm)   = comm;
(app->tstart) = tstart;
(app->tstop)  = tstop;
(app->ntime)  = ntime;
```

then the data structure definitions and wrapper routines are passed to Warp

```
warp_Core  core;
warp_Init(MPI_COMM_WORLD, comm, tstart, tstop, ntime, app,
          my_Phi, my_Init, my_Clone, my_Free, my_Sum, my_Dot, my_Write,
          my_BufSize, my_BufPack, my_BufUnpack,
          &core);
```

then Warp options can be set

```
warp_SetPrintLevel( core, 1);
warp_SetMaxLevels(core, max_levels);
warp_SetNRelax(core, -1, nrelax);
warp_SetAbsTol(core, tol);
warp_SetCFactor(core, -1, cfactor);
warp_SetMaxIter(core, max_iter);
```

then the simulation is run

```
warp_Drive(core);
```

then we clean up

```
warp_Destroy(core);
```

see drive-01 and drive-0∗ for more extensive examples.

Testing Warp

The best overall test for Warp, is to set MaxLevels to 1 which will carry out a sequential time stepping test. Take the output given to you by your Write function and compare it to output from a non-Warp run. Is everything OK? Repeat this test for multilevel Warp.

To do sanity checks of your data structures and wrapper routines, there are also Warp test functions, which can be easily run. The test routines also take as arguments the app structure, spatial communicator comm_x, a stream like stdout for test output and a time step size to test dt. After these arguments, function pointers to wrapper routines are the rest of the arguments. Some of the tests can return a boolean variable to indicate correctness.

```
/* Test init(), write(), free() */
warp_TestInitWrite( app, comm_x, stdout, dt, my_Init, my_Write, my_Free);

/* Test clone() */
warp_TestClone( app, comm_x, stdout, dt, my_Init, my_Write, my_Free, my_Clone);

/* Test sum() */
warp_TestSum( app, comm_x, stdout, dt, my_Init, my_Write, my_Free, my_Clone, my_Sum);

/* Test dot() */
correct = warp_TestDot( app, comm_x, stdout, dt, my_Init, my_Free, my_Clone, my_Sum, my_Dot);

/* Test bufsize(), bufpack(), bufunpack() */
correct = warp_TestBuf( app, comm_x, stdout, dt, my_Init, my_Free, my_Sum, my_Dot, my_BufSize, my_BufPack, my_Buf

/* Test coarsen and refine */
correct = warp_TestCoarsenRefine(app, comm_x, stdout, 0.0, dt, 2*dt, my_Init,
                      my_Write, my_Free, my_Clone, my_Sum, my_Dot, my_CoarsenInjection,
                      my_Refine);
correct = warp_TestCoarsenRefine(app, comm_x, stdout, 0.0, dt, 2*dt, my_Init,
                     my_Write, my_Free, my_Clone, my_Sum, my_Dot, my_CoarsenBilinear,
                     my_Refine);
```

4 Building Warp

- To specify the compilers, flags and options for your machine, edit makefile.inc. For now, we keep it simple and avoid using configure or cmake.
- To make the library, libwarp.a,
  ```
  $ make
  ```
- To make the examples
  ```
  $ make all
  ```
- The makefile lets you pass some parameters like debug with
  ```
  $ make debug=yes
  ```
  or
  ```
  $ make all debug=yes
  ```
  It would also be easy to add additional parameters, e.g., to compile with insure.
- To set compilers and library locations, look in makefile.inc where you can set up an option for your machine to define simple stuff like
  CC = mpicc MPICC = mpicc MPICXX = mpiCC LFLAGS = -lm

5 Compiling the examples

- Type
  drive-0∗ -help

for instructions on how to run any driver

- To run the examples
  mpirun -np 4 drive-∗ [args]
- drive-01 is the simplest example. It implements a scalar ODE and can be compiled and run with no outside dependencies.
- drive-02 implements the 2D heat equation on a regular grid. You must have `hypre` installed and these variables in the Makefile set correctly
```
HYPRE_DIR = ../../linear_solvers/hypre
HYPRE_FLAGS = -I$(HYPRE_DIR)/include
HYPRE_LIB = -L$(HYPRE_DIR)/lib -lHYPRE
```
- drive-03 implements the 3D heat equation on a regular grid, and assumes hypre is installed just like drive-02.
- drive-05 implements the 2D heat equation on a regular grid, but it uses with spatial coarsening. This allows you to use explicit time stepping on each Warp level, regardless of time step size. It assumes hypre is installed just like drive-02.
- drive-04 is sophisticated test bed for various PDEs, mostly parabolic. It relies on the `mfem` package to create general finite element discretizations for the spatial problem. Other packages must be installed in this order.
  - Unpack and install `Metis`
  - Unpack and install hypre
  - Unpack and install mfem. Make the serial version of mfem first by only typing `make`. Then make sure to set these variables correctly in the Makefile
    USE_METIS_5 = YES HYPRE_DIR = where_ever_linear_solvers_is/hypre
  - Make `GLVIS`, which needs serial mfem.
  - Go back to mfem
```
make clean
make parallel
```
  - Got to warp/examples and set these Makefile variables, and them `make drive-04`.
```
METIS_DIR = ../../metis-5.1.0/lib
MFEM_DIR = ../../mfem
MFEM_FLAGS = -I$(MFEM_DIR)
MFEM_LIB = -L$(MFEM_DIR) -lmfem -L$(METIS_DIR) -lmetis
```
- To run drive-04 and glvis, open two windows. In one, start a glvis session
```
./glvis
```
  Then, in the other window, run drive-04
```
mpirun -np ... drive-04 [args]
```

6 Coding Style

Code should follow the `ellemtel` style. See warp/misc/sample_c_code.c, and for emacs and vim style files, see warp/misc/sample.vimrc, and warp/misc/sample.emacs.

7 Using Doxygen

To build the documentation, doxygen must be version 1.8 or greater. Warp documentation uses a markdown syntax both in source file comments and in ∗.md files.

To make the documentation,

```
$ make user_manual
$ acroread user_manual.pdf
```

or to make a more extensive reference manual for developers,

```
$ make developer_manual
$ acroread developer_manual.pdf
```

Developers can run doxygen from a precompiled binary, which may or may not work for your machine,

```
/usr/casc/hypre/warp/share/doxygen/bin/doxygen
```

or build doxygen from

```
/usr/casc/hypre/warp/share/doxygen.tgz
```

- Compiling doxygen requires a number of dependencies like Bison, GraphViz and Flex. Configure will tell you what you're missing
- Unpack doxygen.tgz, then from the doxygen directory
  ```
  ./configure --prefix some_dir_in_your_path
  make
  make install
  ```

Documentation Strategy

- The doxygen comments are to be placed in the header files.
- A sample function declaration using the documenation approach using markdown (including typesetting equations) is in warp.h for the function warp_Init()
- A sample structure documentation is in _warp.h for _warp_Core_struct
- Descriptors for files can also be added, as at the top of warp.h
- The Doxygen manual is at http://www.stack.nl/~dimitri/doxygen/manual/index.html

Warp Doxygen details

The latex manuals are built according to

- docs/local_doxygen.sty
  - Latex style file used
- docs/user_manual_header.tex
  - User manual title page and header info
- docs/developer_manual_header.tex
  - Developer manual title page and header info
- filename.md
  - Extra material in markdown format, like Abstract.md and Introduction.md

- docs/user_manual.conf
  - Doxygen configure file for the user manual
  - The FILE_NAMES tag filters to only include the user interface routines in warp.h
  - The INPUT tag orders the processing of the files and hence the section ordering
- docs/reference_manual.conf
  - Same as user_manual.conf, out the FILE_NAMES tag doesn't exclude anything
- docs/img
  - Contains the images
- To regenerate generic doxygen latex files, type
  ```
  $ doxygen -w latex header.tex footer.tex doxygen.sty doxy.conf
  ```
  The .conf file must then be changed to use the new header file and to copy the local_doxygen.sty file to the latex directory.

8 Regression Testing

Overview

- There are three levels in the testing framework. At each level, the fine-grain output from a `testscript.sh` is dumped into a directory `testscript.dir`, with the standard out and error stored in `testscript.out` and `testscript.err`. The test `testscript.sh` passes if `testscript.err` is empty (nothing is written to standard error).
- Basic instructions:
  ```
  $ ./test.sh diffusion2D.sh
  ```
  Then, see if `diffusion2D.err` is of size 0. If it is not, look at it's contents to see which test failed.
- To add a new regression test: Create a new low level script like `diffusion2D.sh` and then call it from a machine script at level 2.
- Regression tests should be run before pushing code.

Lowest Level Test Scripts

Files used:

- `test.sh`
- `diffusion2D.sh`
- `diffusion2D.saved`

Output:

- `diffusion2D.dir`
- `diffusion2D.err`
- `diffusion2D.out`

At this level, we execute

```
$ ./test.sh diffusion2D.sh
```

or just

```
$ ./diffusion2D.sh
```

The script `diffusion2D.sh` must create `diffusion2D.dir` and place all fine-grain test output in this directory. `test.sh` captures the standard out and error in `diffusion2D.out` and `diffusion2D.err`. The test `diffusion2D.sh` passes if `diffusion2D.err` is empty (nothing is written to standard error).

The strategy for low level scripts like `diffusion2D.sh` is to run a sequence of test drivers such as

```
$ mpirun -np 1 ../examples/drive-02 -pgrid 1 1 1 -nt 256
$ mpirun -np 4 ../examples/drive-02 -pgrid 1 1 4 -nt 256
```

The output from the first mpirun test must then be written to files named

```
diffusion2D.dir/unfiltered.std.out.0
diffusion2D.dir/std.out.0
diffusion2D.dir/std.err.0
```

and the second mpirun test similarly writes the files

```
diffusion2D.dir/unfiltered.std.out.1
diffusion2D.dir/std.out.1
diffusion2D.dir/std.err.1
```

Subsequent tests are written to higher numbered files. The `unfiltered.std.out.num` file contains all of the standard out for the test, while `std.out.num` contains filtered output (usually from a `grep` command) and could contain the output lines such as iteration numbers and number of levels. The file `std.err.num` contains the standard error output.

To see if a test ran correctly, `std.out.num` is compared to saved output in `diffusion2D.saved`. The file `diffusion2D.saved` contains the concatenated output from all the tests that `diffusion2D.sh` will run. For the above example, this file could look like

```
# Begin Test 1
number of levels     = 6
iterations           = 16
# Begin Test 2
number of levels     = 4
iterations           = 8
```

This saved output is split into an individual file for each test (using `# Begin Test` as a delimiter) and these new files are placed in `diffusion2D.dir`. So, after running these two regression tests, `diffusion2D.dir` will contain

```
diffusion2D.saved.0
diffusion2D.saved.1
unfiltered.std.out.0
std.out.0
std.err.0
unfiltered.std.out.1
std.out.1
std.err.1
```

An individual test has passed if `std.err.num` is empty. The file `std.err.num` contains a diff between `diffusion2D.save.num` and `std.out.num` (ignoring whitespace and the delimiter `# Begin Test`).

Last in the directy where you ran `./test.sh diffusion2d.sh`, the files

```
diffusion2D.err
diffusion2D.out
```

will be created. If all the tests passed then `diffusion2D.err` will be empty. Otherwise, it will contain the filenames of the `std.err.num` files that are non-empty, representing failed tests.

Level 2 Scripts

Files used:

- machine-tux.sh

Output:

- machine-tux.dir
- machine-tux.err (only generated if autotest.sh is used to run machine-tux.sh)
- machine-tux.out (only generated if autotest.sh is used to run machine-tux.sh)

At this level, we execute

```
./machine-tux.sh
```

The autotest framework (autotest.sh) calls machine scripts in this way. Each machine script should be short and call lower-level scripts like diffusion2D.sh. The output from lower-level scripts must be moved to machine-tux.dir like this:

```
$ ./test.sh diffusion2D.sh
$ mv -f diffusion2D.dir machine-tux.dir
$ mv -f diffusion2D.out machine-tux.dir
$ mv -f diffusion2D.err machine-tux.dir
```

All error files from diffusion2D.sh will be placed in machine-tux.dir, so if machine-tux.dir has all zero *.err files, then the machine-tux test has passed.

To begin testing on a new machine, like vulcan, add a new machine script similar to machine-tux.sh and change autotest.sh to recognize and run the new machine. To then use autotest.sh with the machine script, you'll have to set up a passwordless connection from the new machine to

```
/usr/casc/hypre/warp/testing
```

Level 3 Script

Files used:

- autotest.sh

Output:

- test/autotest_finished
- /usr/casc/hypre/warp/testing/AUTOTEST-20**.**.**-Day

At the highest level sits autotest.sh and is called automatically as a cronjob. If you just want to check to see if you've broken anything with a commit, just use lower level scripts.

There are four steps to running autotest.

- Step 1
  ```
  $ ./autotesh.sh -init
  ```
  will do a pull from master for the current working repository and recompile warp. The autotest output files (autotest.err and autotest.out) and the output directory (autotest_finished) are initialized.
- Step 2
  ```
  $ ./autotest.sh -tux343
  ```

will run the autotests on tux343. This command will look for a `machine-tux.sh`, and execute it, moving the resulting

```
machine-tux.dir
machine-tux.err
machine-tux.out
```

into `test/autotest_finished`.

- Step 3

```
$ ./autotest.sh -remote-copy
```

will copy `/test/autotest_finished/*` to a time-stamped directory such as `/usr/casc/hypre/warp/testing/-AUTOTEST-2013.11.18-Mon`

```
$ ./autotesh.sh -remote-copy tux343
```

will ssh through tux343 to copy to `/usr/casc`. Multiple machines may independently be running regression tests and then copy to `AUTOTEST-2013.11.18-Mon`.

- Step 4

```
$ ./autotest.sh -summary-email
```

will email everyone listed in in `$email_list` in `autotesh.sh`.

Cronfile

To add entries to your crontab, put your new cronjob lines in cronfile. Then see what you already have in your crontab file with

```
$ crontab -l
```

Next, append to cronfile whatever you already have $ crontab -l >> cronfile

Finally, tell crontab to use your cronfile

```
$ crontab cronfile
```

Then make sure it took affect with

```
$ crontab -l
```

Crontab entry format uses '*' to mean "every" and '*/m' to mean "every m-th". The first five entries on each line correspond respectively to:

- minute (0-56)
- hour (0-23)
- day of month (1-31)
- month (1-12)
- day of week (0-6)(0=Sunday)

Jacob's crontab (on tux343):

```
05 14 * * * source /etc/profile; source $HOME/.bashrc; cd $HOME/joint_repos/warp/test; ./autotest.sh -init
05 14 * * * source /etc/profile; source $HOME/.bashrc; cd $HOME/joint_repos/warp/test; ./autotest.sh -tux343
05 14 * * * source /etc/profile; source $HOME/.bashrc; cd $HOME/joint_repos/warp/test; ./autotest.sh -remote-co
05 14 * * * source /etc/profile; source $HOME/.bashrc; cd $HOME/joint_repos/warp/test; ./autotest.sh -summary-e
```

9 Data Structure Index

9.1 Data Structures

Here are the data structures with brief descriptions:

10 File Index

10.1 File List

Here is a list of all files with brief descriptions:

11 Data Structure Documentation

11.1 _warp_AccuracyHandle Struct Reference

Data Fields

- warp_Int matchF
- warp_Real value
- warp_Real old_value
- warp_Real loose
- warp_Real tight
- warp_Int tight_used

11.1.1 Field Documentation

11.1.1.1 warp_Real loose

11.1.1.2 warp_Int matchF

11.1.1.3 warp_Real old_value

11.1.1.4 warp_Real tight

11.1.1.5 warp_Int tight_used

11.1.1.6 warp_Real value

The documentation for this struct was generated from the following file:

- _warp.h

11.2 _warp_CommHandle Struct Reference

Data Fields

- warp_Int request_type
- warp_Int num_requests
- MPI_Request * requests
- MPI_Status * status
- void * buffer
- warp_Vector * vector_ptr

11.2.1 Field Documentation

11.2.1.1 void* buffer

11.2.1.2 warp_Int num_requests

11.2.1.3 warp_Int request_type

11.2.1.4 MPI_Request* requests

11.2.1.5 MPI_Status* status

11.2.1.6 warp_Vector* vector_ptr

The documentation for this struct was generated from the following file:

- _warp.h

11.3 _warp_Core Struct Reference

Data Fields

- MPI_Comm comm_world
- MPI_Comm comm
- warp_Real tstart
- warp_Real tstop
- warp_Int ntime

- warp_App app
- warp_PtFcnPhi phi
- warp_PtFcnInit init
- warp_PtFcnClone clone
- warp_PtFcnFree free
- warp_PtFcnSum sum
- warp_PtFcnDot dot
- warp_PtFcnWrite write
- warp_PtFcnBufSize bufsize
- warp_PtFcnBufPack bufpack
- warp_PtFcnBufUnpack bufunpack
- warp_PtFcnCoarsen coarsen
- warp_PtFcnRefine refine
- warp_Int write_level
- warp_Int print_level
- warp_Int max_levels
- warp_Int max_coarse
- warp_Real tol
- warp_Int rtol
- warp_Int * nrels
- warp_Int nrdefault
- warp_Int * cfactors
- warp_Int cfdefault
- warp_Int max_iter
- warp_Int niter
- warp_Real rnorm
- warp_Int fmg
- warp_Int nfmg_Vcyc
- _warp_AccuracyHandle * accuracy
- warp_Int gupper
- warp_Int * rfactors
- warp_Int nlevels
- _warp_Grid ** grids
- warp_Real localtime
- warp_Real globaltime

11.3.1 Detailed Description

The typedef _warp_Core struct is a **critical** part of warp and is passed to *each* routine in warp. It thus allows each routine access to the basic warp attributes such as the user's

- phi (ϕ) function where

  $$\phi(\mathbf{u}_i) = \mathbf{u}_{i+1}$$

  evolves the solution
- warp_App structure

11.3.2 Field Documentation

11.3.2.1 _warp_AccuracyHandle* accuracy

accuracy of spatial solves on different levels

11.3.2.2 warp_App app

application data for the user

11.3.2.3 warp_PtFcnBufPack bufpack

pack a buffer

11.3.2.4 warp_PtFcnBufSize bufsize

return buffer size

11.3.2.5 warp_PtFcnBufUnpack bufunpack

unpack a buffer

11.3.2.6 warp_Int* cfactors

coarsening factors

11.3.2.7 warp_Int cfdefault

default coarsening factor

11.3.2.8 warp_PtFcnClone clone

clone a vector

11.3.2.9 warp_PtFcnCoarsen coarsen

(optional) return a coarsened vector

11.3.2.10 MPI_Comm comm

communicator for the time dimension

11.3.2.11 MPI_Comm comm_world

11.3.2.12 warp_PtFcnDot dot

dot product

11.3.2.13 warp_Int fmg

use FMG cycle

11.3.2.14 warp_PtFcnFree free

free up a vector

11.3.2.15 warp_Real globaltime

global wall time for warp_Drive()

11.3.2.16 _warp_Grid** grids

pointer to temporal grid structures for each level

11.3.2.17 warp_Int gupper

global upper index on the fine grid

11.3.2.18 warp_PtFcnInit init

return an initial solution vector

11.3.2.19 warp_Real localtime

local wall time for warp_Drive()

11.3.2.20 warp_Int max_coarse

maximum allowed coarse grid size (in terms of C-points)

11.3.2.21 warp_Int max_iter

maximum number of multigrid in time iterations

11.3.2.22 warp_Int max_levels

maximum number of temporal grid levels

11.3.2.23 warp_Int nfmg_Vcyc

number of V-cycle calls at each level in FMG

11.3.2.24 warp_Int niter

number of iterations

11.3.2.25 warp_Int nlevels

number of temporal grid levels

11.3.2.26 warp_Int nrdefault

default number of pre-relaxations

11.3.2.27 warp_Int* nrels

number of pre-relaxations on each level

11.3.2.28 warp_Int ntime

initial number of time intervals

11.3.2.29 warp_PtFcnPhi phi

apply phi function

11.3.2.30 warp_Int print_level

determines amount of output printed to screem (0,1,2)

11.3.2.31 warp_PtFcnRefine refine

(optional) return a refined vector

11.3.2.32 warp_Int* rfactors

refinement factors for finest grid (if any)

11.3.2.33 warp_Real rnorm

residual norm

11.3.2.34 warp_Int rtol

use relative tolerance

11.3.2.35 warp_PtFcnSum sum

vector sum

11.3.2.36 warp_Real tol

stopping tolerance

11.3.2.37 warp_Real tstart

start time

11.3.2.38 warp_Real tstop

stop time

11.3.2.39 warp_PtFcnWrite write

write the vector

11.3.2.40 warp_Int write_level

determines how often to call the user's write routine

The documentation for this struct was generated from the following file:

- _warp.h

11.4 _warp_Grid Struct Reference

Data Fields

- warp_Int level
- warp_Int ilower
- warp_Int iupper
- warp_Int clower
- warp_Int cupper
- warp_Int cfactor
- warp_Int ncpoints
- warp_Vector * ua

- warp_Real * ta
- warp_Vector * va
- warp_Vector * wa
- warp_Int recv_index
- warp_Int send_index
- _warp_CommHandle * recv_handle
- _warp_CommHandle * send_handle
- warp_Vector * ua_alloc
- warp_Real * ta_alloc
- warp_Vector * va_alloc
- warp_Vector * wa_alloc

11.4.1 Field Documentation

11.4.1.1 warp_Int cfactor

11.4.1.2 warp_Int clower

11.4.1.3 warp_Int cupper

11.4.1.4 warp_Int ilower

11.4.1.5 warp_Int iupper

11.4.1.6 warp_Int level

11.4.1.7 warp_Int ncpoints

11.4.1.8 _warp_CommHandle* recv_handle

11.4.1.9 warp_Int recv_index

11.4.1.10 _warp_CommHandle* send_handle

11.4.1.11 warp_Int send_index

11.4.1.12 warp_Real* ta

11.4.1.13 warp_Real* ta_alloc

11.4.1.14 warp_Vector* ua

11.4.1.15 warp_Vector* ua_alloc

11.4.1.16 warp_Vector* va

11.4.1.17 warp_Vector* va_alloc

11.4.1.18 warp_Vector* wa

11.4.1.19 warp_Vector* wa_alloc

The documentation for this struct was generated from the following file:

- _warp.h

11.5 _warp_Status Struct Reference

Data Fields

- warp_Int iter
- warp_Int level
- warp_Real rnorm
- warp_Int done

11.5.1 Detailed Description

Points to the status structure which defines the status of Warp at a given instant on a some level during a run. The user accesses it through warp_Get∗∗Status() functions.

11.5.2 Field Documentation

11.5.2.1 warp_Int done

11.5.2.2 warp_Int iter

11.5.2.3 warp_Int level

11.5.2.4 warp_Real rnorm

The documentation for this struct was generated from the following file:

- _warp.h

11.6 advection_setup Struct Reference

Data Fields

- int n_fine
- double h_fine
- double dt_fine
- double amp
- double ph
- double om
- int pnr
- int taylorbc
- int nb
- int wb
- double_array_2d ∗ bop_
- double_array_2d ∗ bope_
- double gh
- int nb2
- int wb2
- double_array_2d ∗ bop2_
- double_array_1d ∗ iop2_
- double gh2

- double bder [7]
- double L
- double c_coeff
- double nu_coeff
- double betapcoeff
- double restr_coeff
- double ad_coeff
- int_array_1d * bcnr_
- double_array_1d * alpha_
- double_array_1d * beta_
- int spatial_order
- int warpMaxIter
- double warpResidualLevel
- int copy_level
- grid_fcn * sol_copy
- double t_copy
- int write
- double tstart
- double tstop
- int nsteps

11.6.1 Field Documentation

11.6.1.1 double ad_coeff

11.6.1.2 double_array_1d* alpha_

11.6.1.3 double amp

11.6.1.4 int_array_1d* bcnr_

11.6.1.5 double bder[7]

11.6.1.6 double_array_1d * beta_

11.6.1.7 double betapcoeff

11.6.1.8 double_array_2d* bop2_

11.6.1.9 double_array_2d* bop_

11.6.1.10 double_array_2d* bope_

11.6.1.11 double c_coeff

11.6.1.12 int copy_level

11.6.1.13 double dt_fine

11.6.1.14 double gh

11.6.1.15 double gh2

11.6.1.16 double h_fine

11.6.1.17 double_array_1d* iop2_

11.6.1.18 double L

11.6.1.19 int n_fine

11.6.1.20 int nb

11.6.1.21 int nb2

11.6.1.22 int nsteps

11.6.1.23 double nu_coeff

11.6.1.24 double om

11.6.1.25 double ph

11.6.1.26 int pnr

11.6.1.27 double restr_coeff

11.6.1.28 grid_fcn* sol_copy

11.6.1.29 int spatial_order

11.6.1.30 double t_copy

11.6.1.31 int taylorbc

11.6.1.32 double tstart

11.6.1.33 double tstop

11.6.1.34 int warpMaxIter

11.6.1.35 double warpResidualLevel

11.6.1.36 int wb

11.6.1.37 int wb2

11.6.1.38 int write

The documentation for this struct was generated from the following file:

- advect_data.h

11.7 grid_fcn Struct Reference

Data Fields

- double * sol
- int n
- double h
- double_array_1d * vsol_

11.7.1 Field Documentation

11.7.1.1 double h

11.7.1.2 int n

11.7.1.3 double* sol

11.7.1.4 double_array_1d* vsol_

The documentation for this struct was generated from the following file:

- advect_data.h

12 File Documentation

12.1 _warp.h File Reference

Data Structures

- struct _warp_Status
- struct _warp_CommHandle
- struct _warp_AccuracyHandle
- struct _warp_Grid
- struct _warp_Core

Macros

- #define _warp_CommHandleElt(handle, elt) ((handle) -> elt)
- #define _warp_GridElt(grid, elt) ((grid) -> elt)
- #define _warp_StatusElt(status, elt) ((status) -> elt)
- #define _warp_CoreElt(core, elt) ((core) -> elt)
- #define _warp_CoreFcn(core, fcn) (*((core) -> fcn))
- #define _warp_TAlloc(type, count) ((type *)malloc((size_t)(sizeof(type) * (count))))
- #define _warp_CTAlloc(type, count) ((type *)calloc((size_t)(count), (size_t)sizeof(type)))
- #define _warp_TReAlloc(ptr, type, count) ((type *)realloc((char *)ptr, (size_t)(sizeof(type) * (count))))
- #define _warp_TFree(ptr) (free((char *)ptr), ptr = NULL)
- #define _warp_MapFineToCoarse(findex, cfactor, cindex) (cindex = (findex)/(cfactor))
- #define _warp_MapCoarseToFine(cindex, cfactor, findex) (findex = (cindex)*(cfactor))
- #define _warp_IsFPoint(index, cfactor) ((index)%(cfactor))
- #define _warp_IsCPoint(index, cfactor) (!_warp_IsFPoint(index, cfactor))

Functions

- warp_Int _warp_GetDistribution (warp_Core core, warp_Int *ilower_ptr, warp_Int *iupper_ptr)
- warp_Int _warp_GetProc (warp_Core core, warp_Int level, warp_Int index, warp_Int *proc_ptr)
- warp_Int _warp_CommRecvInit (warp_Core core, warp_Int level, warp_Int index, warp_Vector *vector_ptr, _warp_CommHandle **handle_ptr)
- warp_Int _warp_CommSendInit (warp_Core core, warp_Int level, warp_Int index, warp_Vector vector, _warp_CommHandle **handle_ptr)

- warp_Int _warp_CommWait (warp_Core core, _warp_CommHandle **handle_ptr)
- warp_Int _warp_UCommInit (warp_Core core, warp_Int level)
- warp_Int _warp_UCommInitF (warp_Core core, warp_Int level)
- warp_Int _warp_UCommWait (warp_Core core, warp_Int level)
- warp_Int _warp_UGetInterval (warp_Core core, warp_Int level, warp_Int interval_index, warp_Int *flo_ptr, warp_Int *fhi_ptr, warp_Int *ci_ptr)
- warp_Int _warp_UGetVectorRef (warp_Core core, warp_Int level, warp_Int index, warp_Vector *u_ptr)
- warp_Int _warp_USetVectorRef (warp_Core core, warp_Int level, warp_Int index, warp_Vector u)
- warp_Int _warp_UGetVector (warp_Core core, warp_Int level, warp_Int index, warp_Vector *u_ptr)
- warp_Int _warp_USetVector (warp_Core core, warp_Int level, warp_Int index, warp_Vector u)
- warp_Int _warp_UWriteVector (warp_Core core, warp_Int level, warp_Int index, warp_Status status, warp_Vector u)
- warp_Int _warp_Phi (warp_Core core, warp_Int level, warp_Int index, warp_Real accuracy, warp_Vector u, warp_Int *rfactor)
- warp_Int _warp_Step (warp_Core core, warp_Int level, warp_Int index, warp_Real accuracy, warp_Vector u)
- warp_Int _warp_Coarsen (warp_Core core, warp_Int level, warp_Int f_index, warp_Int c_index, warp_Vector fvector, warp_Vector *cvector)
- warp_Int _warp_Refine (warp_Core core, warp_Int level, warp_Int f_index, warp_Int c_index, warp_Vector cvector, warp_Vector *fvector)
- warp_Int _warp_GridInit (warp_Core core, warp_Int level, warp_Int ilower, warp_Int iupper, _warp_Grid **grid_ptr)
- warp_Int _warp_GridDestroy (warp_Core core, _warp_Grid *grid)
- warp_Int _warp_InitGuess (warp_Core core, warp_Int level)
- warp_Int _warp_CFRelax (warp_Core core, warp_Int level)
- warp_Int _warp_FRestrict (warp_Core core, warp_Int level, warp_Int iter, warp_Real *rnorm_ptr)
- warp_Int _warp_FInterp (warp_Core core, warp_Int level, warp_Int iter, warp_Real rnorm)
- warp_Int _warp_FRefine (warp_Core core, warp_Int *refined_ptr)
- warp_Int _warp_FWrite (warp_Core core, warp_Real rnorm, warp_Int iter, warp_Int level, warp_Int done)
- warp_Int _warp_InitHierarchy (warp_Core core, _warp_Grid *fine_grid)
- warp_Int _warp_InitStatus (warp_Real rnorm, warp_Int iter, warp_Int level, warp_Int done, warp_Status *status_ptr)
- warp_Int _warp_DestroyStatus (warp_Status status)

Variables

- warp_Int _warp_error_flag
- FILE * _warp_printfile

12.1.1 Detailed Description

Define headers for developer routines. This file contains the headers for developer routines.

12.1.2 Macro Definition Documentation

12.1.2.1 #define _warp_CommHandleElt(*handle, elt*) ((handle) -> elt)

12.1.2.2 #define _warp_CoreElt(*core, elt*) ((core) -> elt)

12.1.2.3 #define _warp_CoreFcn(*core, fcn*) (*((core) -> fcn))

12.1.2.4 #define _warp_CTAlloc(*type, count*) ((type *)calloc((size_t)(count), (size_t)sizeof(type)))

12.1.2.5 #define _warp_GridElt(*grid, elt*) ((grid) -> elt)

12.1.2.6 #define _warp_IsCPoint(*index, cfactor*) (!**_warp_IsFPoint**(index, cfactor))

12.1.2.7 #define _warp_IsFPoint(*index, cfactor*) ((index)%(cfactor))

12.1.2.8 #define _warp_MapCoarseToFine(*cindex, cfactor, findex*) (findex = (cindex)*(cfactor))

12.1.2.9 #define _warp_MapFineToCoarse(*findex, cfactor, cindex*) (cindex = (findex)/(cfactor))

12.1.2.10 #define _warp_StatusElt(*status, elt*) ((status) -> elt)

12.1.2.11 #define _warp_TAlloc(*type, count*) ((type *)malloc((size_t)(sizeof(type) * (count))))

12.1.2.12 #define _warp_TFree(*ptr*) (free((char *)ptr), ptr = NULL)

12.1.2.13 #define _warp_TReAlloc(*ptr, type, count*) ((type *)realloc((char *)ptr, (size_t)(sizeof(type) * (count))))

12.1.3 Function Documentation

12.1.3.1 **warp_Int** _warp_CFRelax (**warp_Core** *core,* **warp_Int** *level*)

Do nu sweeps of F-then-C relaxation

12.1.3.2 **warp_Int** _warp_Coarsen (**warp_Core** *core,* **warp_Int** *level,* **warp_Int** *f_index,* **warp_Int** *c_index,* **warp_Vector** *fvector,* **warp_Vector** * *cvector*)

Coarsen in space

12.1.3.3 **warp_Int** _warp_CommRecvInit (**warp_Core** *core,* **warp_Int** *level,* **warp_Int** *index,* **warp_Vector** * *vector_ptr,* **_warp_CommHandle** ** *handle_ptr*)

Blah..

12.1.3.4 **warp_Int** _warp_CommSendInit (**warp_Core** *core,* **warp_Int** *level,* **warp_Int** *index,* **warp_Vector** *vector,* **_warp_CommHandle** ** *handle_ptr*)

Blah..

12.1.3.5 **warp_Int** _warp_CommWait (**warp_Core** *core,* **_warp_CommHandle** ** *handle_ptr*)

Blah..

12.1.3.6 **warp_Int** _warp_DestroyStatus (**warp_Status** *status*)

Destroy a warp_Status structure

12.1.3.7 **warp_Int** _warp_FInterp (**warp_Core** *core,* **warp_Int** *level,* **warp_Int** *iter,* **warp_Real** *rnorm*)

F-Relax on *level* and interpolate to *level-1*

Output:

- The unknown vector *u* on *level* is created by interpolating from *level+1*.
  If set, the user-defined refinement routine is called.

Parameters

core	warp_Core (_warp_Core) struct
level	interp from level to level+1
iter	current iteration number (for user info)
rnorm	residual norm (if level 0)

12.1.3.8 warp_Int _warp_FRefine (warp_Core *core,* warp_Int * *refined_ptr*)

Create a new fine grid based on user refinement factor information, then F-relax and interpolate to the new fine grid and create a new multigrid hierarchy. In general, this will require load re-balancing as well.

RDF: Todo

12.1.3.9 warp_Int _warp_FRestrict (warp_Core *core,* warp_Int *level,* warp_Int *iter,* warp_Real * *rnorm_ptr*)

F-Relax on *level* and then restrict to *level+1*

Output:

- The restricted vectors *va* and *wa* will be created, representing *level+1* versions of the unknown and rhs vectors. If set, the user-defined coarsening routine is called.
- If *level==0*, then *rnorm_ptr* will contain the residual norm.

Parameters

core	warp_Core (_warp_Core) struct
level	restrict from level to level+1
iter	current iteration number (for user info)
rnorm_ptr	pointer to residual norm (if level 0)

12.1.3.10 warp_Int _warp_FWrite (warp_Core *core,* warp_Real *rnorm,* warp_Int *iter,* warp_Int *level,* warp_Int *done*)

Write out the solution on grid level

12.1.3.11 warp_Int _warp_GetDistribution (warp_Core *core,* warp_Int * *ilower_ptr,* warp_Int * *iupper_ptr*)

Determine processor distribution. This must agree with GetProc().

12.1.3.12 warp_Int _warp_GetProc (warp_Core *core,* warp_Int *level,* warp_Int *index,* warp_Int * *proc_ptr*)

Returns -1 if index is out of range

12.1.3.13 warp_Int _warp_GridDestroy (warp_Core *core,* _warp_Grid * *grid*)

Blah..

12.1.3.14 warp_Int _warp_GridInit (warp_Core *core,* warp_Int *level,* warp_Int *ilower,* warp_Int *iupper,* _warp_Grid ** *grid_ptr*)

Create a new grid object

12.1.3.15 warp_Int _warp_InitGuess (warp_Core *core,* warp_Int *level*)

Set initial guess at C-points

12.1.3.16 **warp_Int _warp_InitHierarchy (warp_Core *core,* _warp_Grid * *fine_grid*)**

Initialize (and re-initialize) hierarchy

12.1.3.17 **warp_Int _warp_InitStatus (warp_Real *rnorm,* warp_Int *iter,* warp_Int *level,* warp_Int *done,* warp_Status * *status_ptr*)**

Initialize a warp_Status structure

12.1.3.18 **warp_Int _warp_Phi (warp_Core *core,* warp_Int *level,* warp_Int *index,* warp_Real *accuracy,* warp_Vector *u,* warp_Int * *rfactor*)**

Apply Phi

12.1.3.19 **warp_Int _warp_Refine (warp_Core *core,* warp_Int *level,* warp_Int *f_index,* warp_Int *c_index,* warp_Vector *cvector,* warp_Vector * *fvector*)**

Refine in space

12.1.3.20 **warp_Int _warp_Step (warp_Core *core,* warp_Int *level,* warp_Int *index,* warp_Real *accuracy,* warp_Vector *u*)**

Integrate one time step

12.1.3.21 **warp_Int _warp_UCommInit (warp_Core *core,* warp_Int *level*)**

Working on all intervals

12.1.3.22 **warp_Int _warp_UCommInitF (warp_Core *core,* warp_Int *level*)**

Working only on F-pt intervals

12.1.3.23 **warp_Int _warp_UCommWait (warp_Core *core,* warp_Int *level*)**

Finish up communication

12.1.3.24 **warp_Int _warp_UGetInterval (warp_Core *core,* warp_Int *level,* warp_Int *interval_index,* warp_Int * *flo_ptr,* warp_Int * *fhi_ptr,* warp_Int * *ci_ptr*)**

Blah..

12.1.3.25 **warp_Int _warp_UGetVector (warp_Core *core,* warp_Int *level,* warp_Int *index,* warp_Vector * *u_ptr*)**

Returns the u-vector on grid 'level' at point 'index'. If 'index' is my "receive index" (as set by UCommInit(), for example), the u-vector will be received from a neighbor processor. If 'index' is within my index range and is also a C-point, the saved value of u will be used. A NULL value is returned otherwise.

12.1.3.26 **warp_Int _warp_UGetVectorRef (warp_Core *core,* warp_Int *level,* warp_Int *index,* warp_Vector * *u_ptr*)**

Returns a reference to the local u-vector on grid 'level' at point 'index'. If 'index' is not a C-point and within my index range, NULL is returned.

12.1.3.27 **warp_Int _warp_USetVector (warp_Core *core,* warp_Int *level,* warp_Int *index,* warp_Vector *u*)**

Sets the u-vector on grid 'level' at point 'index'. If 'index' is my "send index", a send is initiated to a neighbor processor. If 'index' is within my index range and is also a C-point, the value is saved locally.

12.1.3.28 warp_Int _warp_USetVectorRef (warp_Core *core,* warp_Int *level,* warp_Int *index,* warp_Vector *u*)

Stores a reference to the u-vector on grid 'level' at point 'index'. If 'index' is not a C-point and within my index range, nothing is done.

12.1.3.29 warp_Int _warp_UWriteVector (warp_Core *core,* warp_Int *level,* warp_Int *index,* warp_Status *status,* warp_Vector *u*)

Blah..

12.1.4 Variable Documentation

12.1.4.1 warp_Int _warp_error_flag

12.1.4.2 FILE* _warp_printfile

12.2 advect_data.h File Reference

Data Structures

- struct grid_fcn
- struct advection_setup

Macros

- #define MY_EPS 1e-12

Enumerations

- enum bcType { Periodic, Dirichlet, Extrapolation }

Functions

- int init_grid_fcn (advection_setup *kd_, double t, grid_fcn **u_handle)
- void init_advection_solver (double h, double amp, double ph, double om, int pnr, int taylorbc, double L, double cfl, int nstepsset, int nsteps, double tfinal, double wave_speed, double viscosity, int bcLeft, int bcRight, int warpMaxIter, double warpResidualLevel, double restr_coeff, double ad_coeff, int spatial_order, advection_setup *kd_)
- int explicit_rk4_stepper (advection_setup *kd_, double t, double tend, double accuracy, grid_fcn *gf_, int *rfact_)
- int copy_grid_fcn (advection_setup *kd_, grid_fcn *u_, grid_fcn **v_handle)
- int free_grid_fcn (advection_setup *kd_, grid_fcn *u_)
- int sum_grid_fcn (advection_setup *kd_, double alpha, grid_fcn *x_, double beta, grid_fcn *y_)
- int dot_grid_fcn (advection_setup *kd_, grid_fcn *u_, grid_fcn *v_, double *dot_ptr)
- int save_grid_fcn (advection_setup *kd_, double t, warp_Status status, grid_fcn *u_)
- int gridfcn_BufSize (advection_setup *kd_, int *size_ptr)
- int gridfcn_BufPack (advection_setup *kd_, grid_fcn *u_, void *buffer)
- int gridfcn_BufUnpack (advection_setup *kd_, void *buffer, warp_Vector *u_handle)
- int gridfcn_Refine (advection_setup *kd_, double tstart, double f_tminus, double f_tplus, double c_tminus, double c_tplus, grid_fcn *cu_, grid_fcn **fu_handle)
- int gridfcn_Coarsen (advection_setup *kd_, double tstart, double f_tminus, double f_tplus, double c_tminus, double c_tplus, grid_fcn *fu_, grid_fcn **cu_handle)
- void exact1 (grid_fcn *w, double t, advection_setup *kd_)

- void exact_t (grid_fcn ∗w, double t, advection_setup ∗kd_)
- void exact_x (grid_fcn ∗w, double t, advection_setup ∗kd_)
- void exact_xx (grid_fcn ∗w, double t, advection_setup ∗kd_)
- void bdata (grid_fcn ∗w, double t, advection_setup ∗kd_)
- void bop6g (double t, double_array_2d ∗q06_)
- void diffusion_coeff_4 (double_array_1d ∗iop2_, double_array_2d ∗bop2_, double ∗gh2, double bder[5])
- void diffusion_coeff_6 (double_array_1d ∗iop2_, double_array_2d ∗bop2_, double ∗gh2, double bder[7])
- void twbndry1 (double ∗bdataL, double ∗bdataR, int stage, double t, double dt, advection_setup ∗kd_)
- void assign_gp (grid_fcn ∗w, double bdataL, double bdataR, advection_setup ∗kd_)
- void dwdt (grid_fcn ∗w, grid_fcn ∗dwdt, double t, double bdata[2], advection_setup ∗kd_)
- void dwdx (grid_fcn ∗w, grid_fcn ∗dwdt, advection_setup ∗kd_)
- void d2wdx2 (grid_fcn ∗w, grid_fcn ∗wxx, advection_setup ∗kd_)
- void dvdtbndry (grid_fcn ∗w, grid_fcn ∗dwdt, double t, advection_setup ∗kd_)
- void evaldiff (grid_fcn ∗w, grid_fcn ∗we, double ∗l2, double ∗li)
- void evalnorm (grid_fcn ∗w, double ∗l2, double ∗li)
- void adterm (grid_fcn ∗w, grid_fcn ∗dpwdxp, advection_setup ∗kd_)

12.2.1 Macro Definition Documentation

12.2.1.1 #define MY_EPS 1e-12

12.2.2 Enumeration Type Documentation

12.2.2.1 enum bcType

Enumerator

Periodic

Dirichlet

Extrapolation

12.2.3 Function Documentation

12.2.3.1 void adterm (grid_fcn ∗ *w,* grid_fcn ∗ *dpwdxp,* advection_setup ∗ *kd_*)

12.2.3.2 void assign_gp (grid_fcn ∗ *w,* double *bdataL,* double *bdataR,* advection_setup ∗ *kd_*)

12.2.3.3 void bdata (grid_fcn ∗ *w,* double *t,* advection_setup ∗ *kd_*)

12.2.3.4 void bop6g (double *t,* double_array_2d ∗ *q06_*)

12.2.3.5 int copy_grid_fcn (advection_setup ∗ *kd_,* grid_fcn ∗ *u_,* grid_fcn ∗∗ *v_handle*)

12.2.3.6 void d2wdx2 (grid_fcn ∗ *w,* grid_fcn ∗ *wxx,* advection_setup ∗ *kd_*)

12.2.3.7 void diffusion_coeff_4 (double_array_1d ∗ *iop2_,* double_array_2d ∗ *bop2_,* double ∗ *gh2,* double *bder[5]*)

12.2.3.8 void diffusion_coeff_6 (double_array_1d ∗ *iop2_,* double_array_2d ∗ *bop2_,* double ∗ *gh2,* double *bder[7]*)

12.2.3.9 int dot_grid_fcn (advection_setup ∗ *kd_,* grid_fcn ∗ *u_,* grid_fcn ∗ *v_,* double ∗ *dot_ptr*)

12.2.3.10 void dvdtbndry (grid_fcn ∗ *w,* grid_fcn ∗ *dwdt,* double *t,* advection_setup ∗ *kd_*)

12.2.3.11 void dwdt (grid_fcn * *w*, grid_fcn * *dwdt*, double *t*, double *bdata[2]*, advection_setup * *kd_*)

12.2.3.12 void dwdx (grid_fcn * *w*, grid_fcn * *dwdt*, advection_setup * *kd_*)

12.2.3.13 void evaldiff (grid_fcn * *w*, grid_fcn * *we*, double * *l2*, double * *li*)

12.2.3.14 void evalnorm (grid_fcn * *w*, double * *l2*, double * *li*)

12.2.3.15 void exact1 (grid_fcn * *w*, double *t*, advection_setup * *kd_*)

12.2.3.16 void exact_t (grid_fcn * *w*, double *t*, advection_setup * *kd_*)

12.2.3.17 void exact_x (grid_fcn * *w*, double *t*, advection_setup * *kd_*)

12.2.3.18 void exact_xx (grid_fcn * *w*, double *t*, advection_setup * *kd_*)

12.2.3.19 int explicit_rk4_stepper (advection_setup * *kd_*, double *t*, double *tend*, double *accuracy*, grid_fcn * *gf_*, int * *rfact_*)

12.2.3.20 int free_grid_fcn (advection_setup * *kd_*, grid_fcn * *u_*)

12.2.3.21 int gridfcn_BufPack (advection_setup * *kd_*, grid_fcn * *u_*, void * *buffer*)

12.2.3.22 int gridfcn_BufSize (advection_setup * *kd_*, int * *size_ptr*)

12.2.3.23 int gridfcn_BufUnpack (advection_setup * *kd_*, void * *buffer*, warp_Vector * *u_handle*)

12.2.3.24 int gridfcn_Coarsen (advection_setup * *kd_*, double *tstart*, double *f_tminus*, double *f_tplus*, double *c_tminus*, double *c_tplus*, grid_fcn * *fu_*, grid_fcn ** *cu_handle*)

12.2.3.25 int gridfcn_Refine (advection_setup * *kd_*, double *tstart*, double *f_tminus*, double *f_tplus*, double *c_tminus*, double *c_tplus*, grid_fcn * *cu_*, grid_fcn ** *fu_handle*)

12.2.3.26 void init_advection_solver (double *h*, double *amp*, double *ph*, double *om*, int *pnr*, int *taylorbc*, double *L*, double *cfl*, int *nstepsset*, int *nsteps*, double *tfinal*, double *wave_speed*, double *viscosity*, int *bcLeft*, int *bcRight*, int *warpMaxIter*, double *warpResidualLevel*, double *restr_coeff*, double *ad_coeff*, int *spatial_order*, advection_setup * *kd_*)

12.2.3.27 int init_grid_fcn (advection_setup * *kd_*, double *t*, grid_fcn ** *u_handle*)

12.2.3.28 int save_grid_fcn (advection_setup * *kd_*, double *t*, warp_Status *status*, grid_fcn * *u_*)

12.2.3.29 int sum_grid_fcn (advection_setup * *kd_*, double *alpha*, grid_fcn * *x_*, double *beta*, grid_fcn * *y_*)

12.2.3.30 void twbndry1 (double * *bdataL*, double * *bdataR*, int *stage*, double *t*, double *dt*, advection_setup * *kd_*)

12.3 c_array.h File Reference

Macros

- #define NO_REAL
- #define compute_index_1d(p, i) p->arrayptr[i-1]
- #define compute_index_2d(p, i, j) p->arrayptr[i-1 + (j-1)*(p->n1)]
- #define compute_index_3d(p, i, j, k) p->arrayptr[(i-1) + (j-1)*(p->n1) + (k-1)*(p->n1)*(p->n2)]
- #define compute_index_4d(p, i, j, k, l)
- #define ARRAY(type)
- #define ARRAY(type)

- #define ARRAY(type)
- #define ARRAY(type)

Functions

- int_array_1d * create_int_array_1d (int n1)
- int_array_2d * create_int_array_2d (int n1, int n2)
- int_array_3d * create_int_array_3d (int n1, int n2, int n3)
- int_array_4d * create_int_array_4d (int n1, int n2, int n3, int n4)
- float_array_1d * create_float_array_1d (int n1)
- float_array_2d * create_float_array_2d (int n1, int n2)
- float_array_3d * create_float_array_3d (int n1, int n2, int n3)
- float_array_4d * create_float_array_4d (int n1, int n2, int n3, int n4)
- double_array_1d * create_double_array_1d (int n1)
- double_array_2d * create_double_array_2d (int n1, int n2)
- double_array_3d * create_double_array_3d (int n1, int n2, int n3)
- double_array_4d * create_double_array_4d (int n1, int n2, int n3, int n4)
- int_array_1d * delete_int_array_1d (int_array_1d *struct_ptr)
- int_array_2d * delete_int_array_2d (int_array_2d *struct_ptr)
- int_array_3d * delete_int_array_3d (int_array_3d *struct_ptr)
- int_array_4d * delete_int_array_4d (int_array_4d *struct_ptr)
- float_array_1d * delete_float_array_1d (float_array_1d *struct_ptr)
- float_array_2d * delete_float_array_2d (float_array_2d *struct_ptr)
- float_array_3d * delete_float_array_3d (float_array_3d *struct_ptr)
- float_array_4d * delete_float_array_4d (float_array_4d *struct_ptr)
- double_array_1d * delete_double_array_1d (double_array_1d *struct_ptr)
- double_array_2d * delete_double_array_2d (double_array_2d *struct_ptr)
- double_array_3d * delete_double_array_3d (double_array_3d *struct_ptr)
- double_array_4d * delete_double_array_4d (double_array_4d *struct_ptr)
- void write_int_array_1d (int_array_1d *struct_ptr, int fd)
- void write_int_array_2d (int_array_2d *struct_ptr, int fd)
- void write_int_array_3d (int_array_3d *struct_ptr, int fd)
- void write_int_array_4d (int_array_4d *struct_ptr, int fd)
- int_array_1d * read_int_array_1d (int fd)
- int_array_2d * read_int_array_2d (int fd)
- int_array_3d * read_int_array_3d (int fd)
- int_array_4d * read_int_array_4d (int fd)
- void ascii_int_array_1d (int_array_1d *struct_ptr, FILE *fp, char *name)
- void ascii_int_array_2d (int_array_2d *struct_ptr, FILE *fp, char *name)
- void ascii_int_array_3d (int_array_3d *struct_ptr, FILE *fp, char *name)
- void ascii_int_array_4d (int_array_4d *struct_ptr, FILE *fp, char *name)
- void print_int_array_1d (int_array_1d *struct_ptr)
- void print_int_array_2d (int_array_2d *struct_ptr)
- void print_int_array_3d (int_array_3d *struct_ptr)
- void print_float_array_2d (float_array_2d *struct_ptr)
- void print_double_array_2d (double_array_2d *struct_ptr)
- void print_int_3d_element (int_array_3d *struct_ptr, int i, int j, int k)

12.3.1 Macro Definition Documentation

12.3.1.1 #define ARRAY(*type*)

Value:

```
typedef struct { \
  int n1; \
  type *arrayptr; \
} type ## _array_1d;
```

12.3.1.2 #define ARRAY(*type*)

Value:

```
typedef struct { \
  int n1,n2; \
  type *arrayptr; \
} type ## _array_2d;
```

12.3.1.3 #define ARRAY(*type*)

Value:

```
typedef struct { \
  int n1,n2,n3; \
  type *arrayptr; \
} type ## _array_3d;
```

12.3.1.4 #define ARRAY(*type*)

Value:

```
typedef struct { \
  int n1,n2,n3,n4; \
  type *arrayptr; \
} type ## _array_4d;
```

12.3.1.5 #define compute_index_1d(*p, i*) p->arrayptr[i-1]

12.3.1.6 #define compute_index_2d(*p, i, j*) p->arrayptr[i-1 + (j-1)*(p->n1)]

12.3.1.7 #define compute_index_3d(*p, i, j, k*) p->arrayptr[(i-1) + (j-1)*(p->n1) + (k-1)*(p->n1)*(p->n2)]

12.3.1.8 #define compute_index_4d(*p, i, j, k, l*)

Value:

```
p->arrayptr[(i-1) + (j-1)*(p->n1) + (k-1)*(p->n1)*(p->n2) + \
        (l-1)*(p->n1)*(p->n2)*(p->n3)]
```

12.3.1.9 #define NO_REAL

12.3.2 Function Documentation

12.3.2.1 void ascii_int_array_1d (int_array_1d * *struct_ptr,* FILE * *fp,* char * *name*)

12.3.2.2 void ascii_int_array_2d (int_array_2d * *struct_ptr,* FILE * *fp,* char * *name*)

12.3.2.3 void ascii_int_array_3d (int_array_3d * *struct_ptr,* FILE * *fp,* char * *name*)

12.3.2.4 void ascii_int_array_4d (int_array_4d * *struct_ptr,* FILE * *fp,* char * *name*)

12.3.2.5 double_array_1d* create_double_array_1d (int *n1*)

12.3.2.6 double_array_2d* create_double_array_2d (int *n1,* int *n2*)

12.3.2.7 double_array_3d* create_double_array_3d (int *n1,* int *n2,* int *n3*)

12.3.2.8 double_array_4d* create_double_array_4d (int *n1,* int *n2,* int *n3,* int *n4*)

12.3.2.9 float_array_1d* create_float_array_1d (int *n1*)

12.3.2.10 float_array_2d* create_float_array_2d (int *n1,* int *n2*)

12.3.2.11 float_array_3d* create_float_array_3d (int *n1,* int *n2,* int *n3*)

12.3.2.12 float_array_4d* create_float_array_4d (int *n1,* int *n2,* int *n3,* int *n4*)

12.3.2.13 int_array_1d* create_int_array_1d (int *n1*)

12.3.2.14 int_array_2d* create_int_array_2d (int *n1,* int *n2*)

12.3.2.15 int_array_3d* create_int_array_3d (int *n1,* int *n2,* int *n3*)

12.3.2.16 int_array_4d* create_int_array_4d (int *n1,* int *n2,* int *n3,* int *n4*)

12.3.2.17 double_array_1d* delete_double_array_1d (double_array_1d * *struct_ptr*)

12.3.2.18 double_array_2d* delete_double_array_2d (double_array_2d * *struct_ptr*)

12.3.2.19 double_array_3d* delete_double_array_3d (double_array_3d * *struct_ptr*)

12.3.2.20 double_array_4d* delete_double_array_4d (double_array_4d * *struct_ptr*)

12.3.2.21 float_array_1d* delete_float_array_1d (float_array_1d * *struct_ptr*)

12.3.2.22 float_array_2d* delete_float_array_2d (float_array_2d * *struct_ptr*)

12.3.2.23 float_array_3d* delete_float_array_3d (float_array_3d * *struct_ptr*)

12.3.2.24 float_array_4d* delete_float_array_4d (float_array_4d * *struct_ptr*)

12.3.2.25 int_array_1d* delete_int_array_1d (int_array_1d * *struct_ptr*)

12.3.2.26 int_array_2d* delete_int_array_2d (int_array_2d * *struct_ptr*)

12.3.2.27 int_array_3d* delete_int_array_3d (int_array_3d * *struct_ptr*)

12.3.2.28 int_array_4d* delete_int_array_4d (int_array_4d * *struct_ptr*)

12.3.2.29 void print_double_array_2d (double_array_2d * *struct_ptr*)

12.3.2.30 void print_float_array_2d (float_array_2d * *struct_ptr*)

12.3.2.31 void print_int_3d_element (int_array_3d * *struct_ptr,* int *i,* int *j,* int *k*)

12.3.2.32 void print_int_array_1d (int_array_1d * *struct_ptr*)

12.3.2.33 void print_int_array_2d (int_array_2d * *struct_ptr*)

12.3.2.34 void print_int_array_3d (int_array_3d * *struct_ptr*)

12.3.2.35 int_array_1d* read_int_array_1d (int *fd*)

12.3.2.36 int_array_2d* read_int_array_2d (int *fd*)

12.3.2.37 int_array_3d* read_int_array_3d (int *fd*)

12.3.2.38 int_array_4d* read_int_array_4d (int *fd*)

12.3.2.39 void write_int_array_1d (int_array_1d * *struct_ptr,* int *fd*)

12.3.2.40 void write_int_array_2d (int_array_2d * *struct_ptr,* int *fd*)

12.3.2.41 void write_int_array_3d (int_array_3d * *struct_ptr,* int *fd*)

12.3.2.42 void write_int_array_4d (int_array_4d * *struct_ptr,* int *fd*)

12.4 util.h File Reference

Functions

- warp_Int _warp_ProjectInterval (warp_Int ilower, warp_Int iupper, warp_Int index, warp_Int stride, warp_Int *pilower, warp_Int *piupper)
- warp_Int _warp_SetAccuracy (warp_Real rnorm, warp_Real loose_tol, warp_Real tight_tol, warp_Real old-Accuracy, warp_Real tol, warp_Real *paccuracy)
- warp_Int _warp_printf (const char *format,...)
- warp_Int _warp_ParFprintfFlush (FILE *file, warp_Int myid, char *message,...)

12.4.1 Detailed Description

Define headers for utility routines. This file contains the headers for utility routines. Essentially, if a routine does not take warp_Core (or other warp specific structs) as an argument, then it's a utility routine.

12.4.2 Function Documentation

12.4.2.1 warp_Int _warp_ParFprintfFlush (FILE * *file,* warp_Int *myid,* char * *message,* ...)

This is a function that allows for "sane" printing of information in parallel. Currently, only myid = 0 prints, but this can be updated as needs change.

Concatenate string1 and string2, print to file, and then flush the file stream. string1 + string2 must be less than 255 characters.

12.4.2.2 warp_Int _warp_printf (const char * *format,* ...)

12.4.2.3 warp_Int _warp_ProjectInterval (warp_Int *ilower,* warp_Int *iupper,* warp_Int *index,* warp_Int *stride,* warp_Int * *pilower,* warp_Int * *piupper*)

12.4.2.4 warp_Int _warp_SetAccuracy (warp_Real *rnorm,* warp_Real *loose_tol,* warp_Real *tight_tol,* warp_Real *oldAccuracy,* warp_Real *tol,* warp_Real * *paccuracy*)

12.5 warp.h File Reference

Typedefs

- typedef int warp_Int
- typedef double warp_Real
- typedef struct _warp_App_struct * warp_App
- typedef struct _warp_Vector_struct * warp_Vector
- typedef struct _warp_Status_struct * warp_Status
- typedef warp_Int(* warp_PtFcnPhi)(warp_App app, warp_Real tstart, warp_Real tstop, warp_Real accuracy, warp_Vector u, warp_Int *rfactor_ptr)
- typedef warp_Int(* warp_PtFcnInit)(warp_App app, warp_Real t, warp_Vector *u_ptr)
- typedef warp_Int(* warp_PtFcnClone)(warp_App app, warp_Vector u, warp_Vector *v_ptr)
- typedef warp_Int(* warp_PtFcnFree)(warp_App app, warp_Vector u)
- typedef warp_Int(* warp_PtFcnSum)(warp_App app, warp_Real alpha, warp_Vector x, warp_Real beta, warp_Vector y)
- typedef warp_Int(* warp_PtFcnDot)(warp_App app, warp_Vector u, warp_Vector v, warp_Real *dot_ptr)
- typedef warp_Int(* warp_PtFcnWrite)(warp_App app, warp_Real t, warp_Status status, warp_Vector u)
- typedef warp_Int(* warp_PtFcnBufSize)(warp_App app, warp_Int *size_ptr)
- typedef warp_Int(* warp_PtFcnBufPack)(warp_App app, warp_Vector u, void *buffer)
- typedef warp_Int(* warp_PtFcnBufUnpack)(warp_App app, void *buffer, warp_Vector *u_ptr)
- typedef warp_Int(* warp_PtFcnCoarsen)(warp_App app, warp_Real tstart, warp_Real f_tminus, warp_Real f_tplus, warp_Real c_tminus, warp_Real c_tplus, warp_Vector fu, warp_Vector *cu_ptr)
- typedef warp_Int(* warp_PtFcnRefine)(warp_App app, warp_Real tstart, warp_Real f_tminus, warp_Real f_tplus, warp_Real c_tminus, warp_Real c_tplus, warp_Vector cu, warp_Vector *fu_ptr)
- typedef struct _warp_Core_struct * warp_Core

Functions

- warp_Int warp_Init (MPI_Comm comm_world, MPI_Comm comm, warp_Real tstart, warp_Real tstop, warp_Int ntime, warp_App app, warp_PtFcnPhi phi, warp_PtFcnInit init, warp_PtFcnClone clone, warp_PtFcnFree free, warp_PtFcnSum sum, warp_PtFcnDot dot, warp_PtFcnWrite write, warp_PtFcnBufSize bufsize, warp_PtFcnBufPack bufpack, warp_PtFcnBufUnpack bufunpack, warp_Core *core_ptr)
- warp_Int warp_Drive (warp_Core core)
- warp_Int warp_Destroy (warp_Core core)
- warp_Int warp_PrintStats (warp_Core core)
- warp_Int warp_SetLoosexTol (warp_Core core, warp_Int level, warp_Real loose_tol)
- warp_Int warp_SetTightxTol (warp_Core core, warp_Int level, warp_Real tight_tol)
- warp_Int warp_SetMaxLevels (warp_Core core, warp_Int max_levels)
- warp_Int warp_SetMaxCoarse (warp_Core core, warp_Int max_coarse)
- warp_Int warp_SetAbsTol (warp_Core core, warp_Real atol)
- warp_Int warp_SetRelTol (warp_Core core, warp_Real rtol)
- warp_Int warp_SetNRelax (warp_Core core, warp_Int level, warp_Int nrelax)
- warp_Int warp_SetCFactor (warp_Core core, warp_Int level, warp_Int cfactor)
- warp_Int warp_SetMaxIter (warp_Core core, warp_Int max_iter)
- warp_Int warp_SetFMG (warp_Core core)

- warp_Int warp_SetNFMGVcyc (warp_Core core, warp_Int nfmg_Vcyc)
- warp_Int warp_SetSpatialCoarsen (warp_Core core, warp_PtFcnCoarsen coarsen)
- warp_Int warp_SetSpatialRefine (warp_Core core, warp_PtFcnRefine refine)
- warp_Int warp_SetPrintLevel (warp_Core core, warp_Int print_level)
- warp_Int warp_SetPrintFile (warp_Core core, const char *printfile_name)
- warp_Int warp_SetWriteLevel (warp_Core core, warp_Int write_level)
- warp_Int warp_SplitCommworld (const MPI_Comm *comm_world, warp_Int px, MPI_Comm *comm_x, MPI_Comm *comm_t)
- warp_Int warp_GetStatusResidual (warp_Status status, warp_Real *rnorm_ptr)
- warp_Int warp_GetStatusIter (warp_Status status, warp_Int *iter_ptr)
- warp_Int warp_GetStatusLevel (warp_Status status, warp_Int *level_ptr)
- warp_Int warp_GetStatusDone (warp_Status status, warp_Int *done_ptr)
- warp_Int warp_GetNumIter (warp_Core core, warp_Int *niter_ptr)
- warp_Int warp_GetRNorm (warp_Core core, warp_Real *rnorm_ptr)

12.5.1 Detailed Description

Define headers for user interface routines. This file contains routines used to allow the user to initialize, run and get and set warp.

12.5.2 Typedef Documentation

12.5.2.1 typedef struct _warp_App_struct* warp_App

Blah...

12.5.2.2 typedef struct _warp_Core_struct* warp_Core

Points to the core structure defined in _warp.h

12.5.2.3 typedef int warp_Int

12.5.2.4 typedef warp_Int(* warp_PtFcnBufPack)(warp_App app, warp_Vector u, void *buffer)

Blah...

12.5.2.5 typedef warp_Int(* warp_PtFcnBufSize)(warp_App app, warp_Int *size_ptr)

Blah...

12.5.2.6 typedef warp_Int(* warp_PtFcnBufUnpack)(warp_App app, void *buffer, warp_Vector *u_ptr)

Blah...

12.5.2.7 typedef warp_Int(* warp_PtFcnClone)(warp_App app, warp_Vector u, warp_Vector *v_ptr)

Blah...

12.5.2.8 typedef warp_Int(* warp_PtFcnCoarsen)(warp_App app, warp_Real tstart, warp_Real f_tminus, warp_Real f_tplus, warp_Real c_tminus, warp_Real c_tplus, warp_Vector fu, warp_Vector *cu_ptr)

(Optional) Blah...

12.5.2.9 typedef warp_Int(* warp_PtFcnDot)(warp_App app, warp_Vector u, warp_Vector v, warp_Real *dot_ptr)

Blah...

12.5.2.10 typedef warp_Int(* warp_PtFcnFree)(warp_App app, warp_Vector u)

Blah...

12.5.2.11 typedef warp_Int(* warp_PtFcnInit)(warp_App app, warp_Real t, warp_Vector *u_ptr)

Blah...

12.5.2.12 typedef warp_Int(* warp_PtFcnPhi)(warp_App app, warp_Real tstart, warp_Real tstop, warp_Real accuracy, warp_Vector u, warp_Int *rfactor_ptr)

Blah...

12.5.2.13 typedef warp_Int(* warp_PtFcnRefine)(warp_App app, warp_Real tstart, warp_Real f_tminus, warp_Real f_tplus, warp_Real c_tminus, warp_Real c_tplus, warp_Vector cu, warp_Vector *fu_ptr)

(Optional) Blah...

12.5.2.14 typedef warp_Int(* warp_PtFcnSum)(warp_App app, warp_Real alpha, warp_Vector x, warp_Real beta, warp_Vector y)

Blah...

12.5.2.15 typedef warp_Int(* warp_PtFcnWrite)(warp_App app, warp_Real t, warp_Status status, warp_Vector u)

Blah...

12.5.2.16 typedef double warp_Real

12.5.2.17 typedef struct _warp_Status_struct* warp_Status

Points to the status structure defined in _warp.h

12.5.2.18 typedef struct _warp_Vector_struct* warp_Vector

Blah...

12.5.3 Function Documentation

12.5.3.1 warp_Int warp_Destroy (warp_Core *core*)

Destroy core.

12.5.3.2 warp_Int warp_Drive (warp_Core *core*)

Integrate in time.

12.5.3.3 warp_Int warp_GetNumIter (warp_Core *core,* warp_Int * *niter_ptr*)

After Drive() finishes, this returns the number of iterations taken

12.5.3.4 warp_Int warp_GetRNorm (warp_Core *core,* warp_Real * *rnorm_ptr*)

After Drive() finishes, this returns the last measured residual norm

12.5.3.5 warp_Int warp_GetStatusDone (warp_Status *status,* warp_Int * *done_ptr*)

Return whether warp is done for the current status object

12.5.3.6 warp_Int warp_GetStatusIter (warp_Status *status,* warp_Int * *iter_ptr*)

Return the iteration for the current status object

12.5.3.7 warp_Int warp_GetStatusLevel (warp_Status *status,* warp_Int * *level_ptr*)

Return the warp level for the current status object

12.5.3.8 warp_Int warp_GetStatusResidual (warp_Status *status,* warp_Real * *rnorm_ptr*)

Return the residual for the current status object

12.5.3.9 warp_Int warp_Init (MPI_Comm *comm_world,* MPI_Comm *comm,* warp_Real *tstart,* warp_Real *tstop,* warp_Int *ntime,* warp_App *app,* warp_PtFcnPhi *phi,* warp_PtFcnInit *init,* warp_PtFcnClone *clone,* warp_PtFcnFree *free,* warp_PtFcnSum *sum,* warp_PtFcnDot *dot,* warp_PtFcnWrite *write,* warp_PtFcnBufSize *bufsize,* warp_PtFcnBufPack *bufpack,* warp_PtFcnBufUnpack *bufunpack,* warp_Core * *core_ptr*)

Create a core object with the required initial data.

Output:

- *core_ptr* will point to the newly created warp_Core structure

Parameters

comm_world	Global communicator for space and time
comm	Communicator for temporal dimension
tstart	start time
tstop	End time
ntime	Initial number of temporal grid values
app	User defined structure to hold *state* information
phi	User time stepping routine to advance state one time value
init	Initialize a warp_Vector function on finest temporal grid
clone	Clone a warp_Vector
free	Free a temporal state warp_Vector
sum	Compute vector sum of two temporal states
dot	Compute dot product between two temporal states
write	*Writes* (file, screen..) upon completion.
bufsize	Computes size for MPI buffer for one
bufpack	Packs MPI buffer to contain one temporal state
bufunpack	Unpacks MPI buffer containing one temporal state

core_ptr	Pointer to warp_Core (_warp_Core) struct

12.5.3.10 warp_Int warp_PrintStats (warp_Core *core*)

Print statistics.

12.5.3.11 warp_Int warp_SetAbsTol (warp_Core *core,* warp_Real *atol*)

Set absolute stopping tolerance.

12.5.3.12 warp_Int warp_SetCFactor (warp_Core *core,* warp_Int *level,* warp_Int *cfactor*)

Set the coarsening factor *cfactor* on grid level *level* (level 0 is the finest grid). The default factor is 2 on all levels. To change the default factor, use *level* = -1.

12.5.3.13 warp_Int warp_SetFMG (warp_Core *core*)

Use FMG cycling.

12.5.3.14 warp_Int warp_SetLoosexTol (warp_Core *core,* warp_Int *level,* warp_Real *loose_tol*)

Set loose stopping tolerance for spatial solves on grid level *level* (level 0 is the finest grid).

12.5.3.15 warp_Int warp_SetMaxCoarse (warp_Core *core,* warp_Int *max_coarse*)

Set max allowed coarse grid size (in terms of C-points)

12.5.3.16 warp_Int warp_SetMaxIter (warp_Core *core,* warp_Int *max_iter*)

Set max number of multigrid iterations.

12.5.3.17 warp_Int warp_SetMaxLevels (warp_Core *core,* warp_Int *max_levels*)

Set max number of multigrid levels.

12.5.3.18 warp_Int warp_SetNFMGVcyc (warp_Core *core,* warp_Int *nfmg_Vcyc*)

Set number of V cycles to use at each FMG level (standard is 1)

12.5.3.19 warp_Int warp_SetNRelax (warp_Core *core,* warp_Int *level,* warp_Int *nrelax*)

Set the number of relaxation sweeps *nrelax* on grid level *level* (level 0 is the finest grid). The default is 1 on all levels. To change the default factor, use *level* = -1.

12.5.3.20 warp_Int warp_SetPrintFile (warp_Core *core,* const char * *printfile_name*)

Set output file for print level messages. Default is stdout.

12.5.3.21 warp_Int warp_SetPrintLevel (warp_Core *core,* warp_Int *print_level*)

Set print level for warp. This controls how much information is printed to standard out.

Level 0: no output Level 1: print typical information like a residual history, number of levels in the Warp hierarchy, and so on. Level 2: level 1 output, plus debug level output.

Default is level 1.

12.5.3.22 warp_Int warp_SetRelTol (warp_Core *core,* warp_Real *rtol*)

Set absolute stopping tolerance.

12.5.3.23 warp_Int warp_SetSpatialCoarsen (warp_Core *core,* warp_PtFcnCoarsen *coarsen*)

Set spatial coarsening routine with user-defined routine. Default is no spatial refinment or coarsening.

12.5.3.24 warp_Int warp_SetSpatialRefine (warp_Core *core,* warp_PtFcnRefine *refine*)

Set spatial refinement routine with user-defined routine. Default is no spatial refinment or coarsening.

12.5.3.25 warp_Int warp_SetTightxTol (warp_Core *core,* warp_Int *level,* warp_Real *tight_tol*)

Set tight stopping tolerance for spatial solves on grid level *level* (level 0 is the finest grid).

12.5.3.26 warp_Int warp_SetWriteLevel (warp_Core *core,* warp_Int *write_level*)

Set write level for warp. This controls how often the user's write routine is called.

Level 0: Never call the user's write routine Level 1: Only call the user's write routine after Warp is finished Level 2: Call the user's write routine every iteration in _warp_FRestrict(), which is during the down-cycle part of a Warp iteration

Default is level 1.

12.5.3.27 warp_Int warp_SplitCommworld (const MPI_Comm * *comm_world,* warp_Int *px,* MPI_Comm * *comm_x,* MPI_Comm * *comm_t*)

Split MPI commworld into comm_x and comm_t, the spatial and temporal communicators

Parameters

comm_world	Global communicator to split
px	Number of processors parallelizing space for a single time step
comm_x	Spatial communicator (written as output)
comm_t	Temporal communicator (written as output)

12.6 warp_test.h File Reference

Functions

- warp_Int warp_TestInitWrite (warp_App app, MPI_Comm comm_x, FILE *fp, warp_Real t, warp_PtFcnInit init, warp_PtFcnWrite write, warp_PtFcnFree free)
- warp_Int warp_TestClone (warp_App app, MPI_Comm comm_x, FILE *fp, warp_Real t, warp_PtFcnInit init, warp_PtFcnWrite write, warp_PtFcnFree free, warp_PtFcnClone clone)
- warp_Int warp_TestSum (warp_App app, MPI_Comm comm_x, FILE *fp, warp_Real t, warp_PtFcnInit init, warp_PtFcnWrite write, warp_PtFcnFree free, warp_PtFcnClone clone, warp_PtFcnSum sum)
- warp_Int warp_TestDot (warp_App app, MPI_Comm comm_x, FILE *fp, warp_Real t, warp_PtFcnInit init, warp_PtFcnFree free, warp_PtFcnClone clone, warp_PtFcnSum sum, warp_PtFcnDot dot)
- warp_Int warp_TestBuf (warp_App app, MPI_Comm comm_x, FILE *fp, warp_Real t, warp_PtFcnInit init, warp_PtFcnFree free, warp_PtFcnSum sum, warp_PtFcnDot dot, warp_PtFcnBufSize bufsize, warp_PtFcnBufPack bufpack, warp_PtFcnBufUnpack bufunpack)
- warp_Int warp_TestCoarsenRefine (warp_App app, MPI_Comm comm_x, FILE *fp, warp_Real t, warp_Real fdt, warp_Real cdt, warp_PtFcnInit init, warp_PtFcnWrite write, warp_PtFcnFree free, warp_PtFcnClone clone, warp_PtFcnSum sum, warp_PtFcnDot dot, warp_PtFcnCoarsen coarsen, warp_PtFcnRefine refine)

- warp_Int warp_TestAll (warp_App app, MPI_Comm comm_x, FILE *fp, warp_Real t, warp_Real fdt, warp_Real cdt, warp_PtFcnInit init, warp_PtFcnFree free, warp_PtFcnClone clone, warp_PtFcnSum sum, warp_PtFcnDot dot, warp_PtFcnBufSize bufsize, warp_PtFcnBufPack bufpack, warp_PtFcnBufUnpack bufunpack, warp_PtFcnCoarsen coarsen, warp_PtFcnRefine refine)

12.6.1 Function Documentation

12.6.1.1 warp_Int warp_TestAll (warp_App *app,* MPI_Comm *comm_x,* FILE * *fp,* warp_Real *t,* warp_Real *fdt,* warp_Real *cdt,* warp_PtFcnInit *init,* warp_PtFcnFree *free,* warp_PtFcnClone *clone,* warp_PtFcnSum *sum,* warp_PtFcnDot *dot,* warp_PtFcnBufSize *bufsize,* warp_PtFcnBufPack *bufpack,* warp_PtFcnBufUnpack *bufunpack,* warp_PtFcnCoarsen *coarsen,* warp_PtFcnRefine *refine*)

Returns 0 if the tests fail Returns 1 if the tests pass Check the log messages to see details of which tests failed.

Parameters

app	User defined App structure
comm_x	Spatial communicator
fp	File pointer (could be stdout or stderr) for log messages
t	Time value to initialize vector
fdt	Fine time step value that you spatially coarsen from
cdt	Coarse time step value that you coarsen to
init	Initialize a warp_Vector function on finest temporal grid
free	Free a temporal state warp_Vector
clone	Clone a temporal state warp_Vector
sum	Compute vector sum of two temporal states
dot	Compute dot product of two temporal states
bufsize	Computes size for MPI buffer for one
bufpack	Packs MPI buffer to contain one temporal state
bufunpack	Unpacks MPI buffer containing one temporal state
coarsen	Spatially coarsen a vector. If null, test is skipped.
refine	Spatially refine a vector. If null, test is skipped.

12.6.1.2 warp_Int warp_TestBuf (warp_App *app,* MPI_Comm *comm_x,* FILE * *fp,* warp_Real *t,* warp_PtFcnInit *init,* warp_PtFcnFree *free,* warp_PtFcnSum *sum,* warp_PtFcnDot *dot,* warp_PtFcnBufSize *bufsize,* warp_PtFcnBufPack *bufpack,* warp_PtFcnBufUnpack *bufunpack*)

Returns 0 if the tests fail Returns 1 if the tests pass Check the log messages to see details of which tests failed.

Parameters

app	User defined App structure
comm_x	Spatial communicator
fp	File pointer (could be stdout or stderr) for log messages
t	Time value to test Buffer routines (used to initialize the vectors
init	Initialize a warp_Vector function on finest temporal grid
free	Free a temporal state warp_Vector
sum	Compute vector sum of two temporal states
dot	Compute dot product of two temporal states
bufsize	Computes size for MPI buffer for one
bufpack	Packs MPI buffer to contain one temporal state
bufunpack	Unpacks MPI buffer containing one temporal state

12.6.1.3 **warp_Int** warp_TestClone (**warp_App** *app,* MPI_Comm *comm_x,* FILE * *fp,* **warp_Real** *t,* **warp_PtFcnInit** *init,* **warp_PtFcnWrite** *write,* **warp_PtFcnFree** *free,* **warp_PtFcnClone** *clone*)

Parameters

app	User defined App structure
comm_x	Spatial communicator
fp	File pointer (could be stdout or stderr) for log messages
t	Time value to test clone with
init	Initialize a warp_Vector function on finest temporal grid
write	Write temporal state warp_Vector (can be NULL for no writing)
free	Free a temporal state warp_Vector
clone	Clone a temporal state warp_Vector

12.6.1.4 warp_Int warp_TestCoarsenRefine (warp_App *app,* MPI_Comm *comm_x,* FILE * *fp,* warp_Real *t,* warp_Real *fdt,* warp_Real *cdt,* warp_PtFcnInit *init,* warp_PtFcnWrite *write,* warp_PtFcnFree *free,* warp_PtFcnClone *clone,* warp_PtFcnSum *sum,* warp_PtFcnDot *dot,* warp_PtFcnCoarsen *coarsen,* warp_PtFcnRefine *refine*)

Returns 0 if the tests fail Returns 1 if the tests pass Check the log messages to see details of which tests failed.

Parameters

app	User defined App structure
comm_x	Spatial communicator
fp	File pointer (could be stdout or stderr) for log messages
t	Time value to initialize vector
fdt	Fine time step value that you spatially coarsen from
cdt	Coarse time step value that you coarsen to
init	Initialize a warp_Vector function on finest temporal grid
write	Write temporal state warp_Vector (can be NULL for no writing)
free	Free a temporal state warp_Vector
clone	Clone a temporal state warp_Vector
sum	Compute vector sum of two temporal states
dot	Compute dot product of two temporal states
coarsen	Spatially coarsen a vector
refine	Spatially refine a vector

12.6.1.5 warp_Int warp_TestDot (warp_App *app,* MPI_Comm *comm_x,* FILE * *fp,* warp_Real *t,* warp_PtFcnInit *init,* warp_PtFcnFree *free,* warp_PtFcnClone *clone,* warp_PtFcnSum *sum,* warp_PtFcnDot *dot*)

Returns 0 if the tests fail Returns 1 if the tests pass Check the log messages to see details of which tests failed.

Parameters

app	User defined App structure
comm_x	Spatial communicator
fp	File pointer (could be stdout or stderr) for log messages
t	Time value to test Dot with (used to initialize the vectors
init	Initialize a warp_Vector function on finest temporal grid
free	Free a temporal state warp_Vector
clone	Clone a temporal state warp_Vector
sum	Compute vector sum of two temporal states
dot	Compute dot product of two temporal states

12.6.1.6 warp_Int warp_TestInitWrite (warp_App *app,* MPI_Comm *comm_x,* FILE * *fp,* warp_Real *t,* warp_PtFcnInit *init,* warp_PtFcnWrite *write,* warp_PtFcnFree *free*)

Parameters

app	User defined App structure
comm_x	Spatial communicator
fp	File pointer (could be stdout or stderr) for log messages
t	Time value to test init with
init	Initialize a warp_Vector function on finest temporal grid
write	Write temporal state warp_Vector (can be NULL for no writing)
free	Free a temporal state warp_Vector

12.6.1.7 warp_Int warp_TestSum (warp_App *app*, MPI_Comm *comm_x*, FILE * *fp*, warp_Real *t*, warp_PtFcnInit *init*, warp_PtFcnWrite *write*, warp_PtFcnFree *free*, warp_PtFcnClone *clone*, warp_PtFcnSum *sum*)

Parameters

app	User defined App structure
comm_x	Spatial communicator
fp	File pointer (could be stdout or stderr) for log messages
t	Time value to test Sum with (used to initialize the vectors
init	Initialize a warp_Vector function on finest temporal grid
write	Write temporal state warp_Vector (can be NULL for no writing)
free	Free a temporal state warp_Vector
clone	Clone a temporal state warp_Vector
sum	Compute vector sum of two temporal states

Index