GeoPar

User’s Guide
and
Reference Manual

Peter Altevogt*
Heidelberg Scientific & Technical Center
(HDSTC)
IBM Deutschland Informationssysteme GmbH
Vangerowstr. 18
69115 Heidelberg
Germany

Werner von Bloh†
Potsdam Institute for Climate Impact Research
(PIK)
Telegrafenberg
14473 Potsdam
Germany

*altevogt@heidelbg.ibm.com
†bloh@pik-potsdam.de
Trademarks and Service Marks

The following terms, denoted by an (*) at their first occurrence in this publication, are trademarks or registered trademarks:

AIX
IBM
RISC System/6000
SP1
SP2
RISC System/6000 Scalable POWERparallel Systems
RS/6000 Scalable POWERparallel Systems
RS/6000 SP
Abstract

The library GeoPar is a library to parallelize algorithms defined on simple cubic lattices (multidimensional arrays in C or Fortran) in two and three dimensions. It provides an automatic distribution of the data on an arbitrary number of processors by associating appropriate sized simple cubic sublattices with the processors ("geometric parallelization" or "domain decomposition"). Furthermore, routines to access the data distributed over the processors, to exchange the data on the boundaries between neighbouring processors as well as for global I/O functions are available.

GeoPar is implemented as a parallel library using the MPI message passing interface and is meant to supplement MPI in providing a convenient parallelization of the above characterized class of applications, not to replace message passing.

In the first release of GeoPar, we have implemented the two dimensional version including the C bindings.
Contents

1 Introduction to GeoPar 5

2 User’s Guide 5
   2.1 Installation of GeoPar 5
   2.2 Compiling and linking with GeoPar 6
   2.3 Example: “Game of Live” 7
      2.3.1 Introduction 7
      2.3.2 Parallelization using GeoPar 8
   2.4 Accessing the lattice 11
   2.5 Updating the boundaries 12
   2.6 I/O-Functions 13
   2.7 Limitations of GeoPar 13
   2.8 Performance considerations 14

3 REFERENCE GUIDE 16
   3.1 Functions listed alphabetically 16
   3.2 Macros listed alphabetically 74

4 Appendix 87
1 Introduction to GeoPar

During the last years, multiprocessor systems of the MIMD type with distributed memory have more and more become the appropriate systems to provide the computational resources for huge computer simulations. Especially the use of standard components like RISC processors used in mass-production of workstations in cluster-like systems (e.g. the IBM RISC Systems/6000 Scalable POWERparallel Systems\(^*\) has drastically reduced the prices of such systems, increased their reliability and has resulted in a many similarities between the architectures of multiprocessor systems provided by various vendors. Besides other reasons, this led to a steady increase in the number of installations of multiprocessor systems of the MIMD type with distributed memory. Although there exist quite a few programming models for the programming of such systems (e.g. the data parallel model, the virtual shared memory model, ...) the message passing model seems to be the most universal, expressive and best performing programming model. Furthermore, the existing MPI standard for message passing provides a portable, efficient application programming interface with a good functionality. However, in many applications based on simple cubic lattices, e.g. Ising-like models in statistical physics or “Daisyworld” models to simulate ecological systems, which are most natural parallelized by geometric parallelization resp. domain decomposition, associating simple cubic sublattices with the processors of the multiprocessor system, frequently the same simple communication patterns occur, e.g. the exchange of data defined on the boundaries of the sublattices between the processors.

The goal of the GeoPar library is to provide simple communication patterns, e.g. to update the data defined on the boundaries on all processors and thus to relieve the user from the (tedious and error prone) task to explicitly implement these simple communications using the MPI message passing interface over and over again. Therefore, GeoPar is not a substitute for message passing, it rather complements the MPI message passing interface for the above specified class of applications.

Currently a tested implementation of GeoPar is supported for IBM 9076 SP2 systems, but there should be no major obstacle to implement GeoPar on other systems supporting the MPI message passing interface.

In the sequel we will first give an introduction to using GeoPar (User’s Guide) including concrete examples and then provide a complete overview over the functionality of GeoPar (Reference Manual).

2 User’s Guide

2.1 Installation of GeoPar

GeoPar is provided in form of the “uuencoded” file GeoPar.uu.
2.2 Compiling and linking with GeoPar

For C programmers

To compile and link code using GeoPar library routines, the makefile may contain the following lines of code:

```bash
CC = mpicc
CFLAGS = -us
GPDIR=/usr/local/GeoPar
GPINCDIR=-I$(GPDIR)
GPLIB=$(GPDIR)/libGP2d.a
example: example.c
   $(CC) $(CFLAGS) $(GPINCDIR) $(GPLIB) example.c -o example
```

We have assumed that GeoPar is installed in the directory /usr/local/GeoPar. The installation directory contains the GP2d.h header file with the ANSI–C prototypes and macros and the library file libGP2d.a. Every application using GeoPar must include this header file, i.e. it must contain the following include directive before accessing a GeoPar function:

```bash
#include <GP2d.h>
```

For Fortran 90 programmers

The makefile may look as follows:

```bash
FC = mpxlf
FFLAGS = -g -O -U
GPDIR=/usr/local/GeoPar
GPINCDIR=-I$(GPDIR)
GPLIB=$(GPDIR)/libGP2d.a
example: example.f
   $(FC) $(FFLAGS) $(GPINCDIR) $(GPLIB) example.f -o example
```

Here the installation directory contains the GP2d.f header file with data declarations. This file must included by:

```bash
#include "GP2d.f"
```

Attention!!! The Fortran 90 interface assumes that the compiler arranges complex data structures in the memory like a C compiler.

Running the program

To execute a program, various environment variables on an IBM RS/6000 SP system must be set. If e.g. the program example requires 4 tasks, we want to use the User Space protocol via the High Performance Switch for communication
and the pool of processors\textsuperscript{1} we want to use is identified by “1”, the parameter list would look as follows (using the Korn shell):

\begin{verbatim}
export MP_PROCS=4
export MP_EUIDVICE=css0
export MP_EUILIB=us
export MP_RMPOOL=1
\end{verbatim}

The parameters may be provided explicitely to the shell via the command line or be part of a shell script. For further details read the user’s guide for MPI/POE???.

2.3 Example: “Game of Life”

2.3.1 Introduction

To demonstrate the use of various GeoPar functions, we implement a simple application, called the “Game of Life”. The “Game of Life” is a cellular automaton defined on a rectangular simple cubic lattice in two dimensions and was introduced by John Conway in 1970. On each site of the lattice there exists a cell having only two states, “dead” and “alive”. The evolution of the states of the cells is determined by two simple rules, which depend only on the state of the cell and it’s neighbors (see, e.g., Fig. 1):

Cell is “alive”: Will remain “alive”, if number of living neighbours is 2 or 3, otherwise will be “dead”.

Cell is “dead”: Will be “alive” if number of living neighbours is exactly 3, otherwise it will remain “dead”.

Implementing the “Game of Life” in C, an adequate type definition for a cell is given by

\begin{verbatim}
typedef enum {DEAD=0,LIVE=1} STATE;
\end{verbatim}

The lattice is then defined as

\begin{verbatim}
STATE lattice[sizex][sizey];
\end{verbatim}

and the update rule for the “Game of Life” may be implemented in C as follows:

\begin{verbatim}
STATE updatestate(const STATE **state,int x1,int x2) {
    int count;
    /*
     * Count living neighbor cells of cell (x1,x2):
     */
    count = state[x1-1][x2-1]+state[x1][x2-1]
\end{verbatim}

\textsuperscript{1}The command \texttt{jm status -P} gives an overview over the defined pools on an IBM RS/6000 SP system.
Figure 1: Two different states of the cellular automaton. In the first case the cell (in the middle) will remain "living", while in the second one the cell will die out.

```
+state[x1+1][x2-1]+state[x1-1][x2 ]
+state[x1+1][x2 ]+state[x1-1][x2+1]
+state[x1 ][x2+1]+state[x1+1][x2+1];

/*
 * Update rules of ‘‘Game of Life’’:
 */
if(state[x1][x2]==LIVE)
 return (count==2 || count==3) ? LIVE : DEAD;
else
 return (count==3) ? LIVE : DEAD;
}
```

A typical configuration of the cellular automaton is shown in Fig. 2.

### 2.3.2 Parallelization using GeoPar

The first step in a parallelization using the GeoPar library is to initialize the parameters of the simulation using the `struct GP2d_InputParams` as defined in GP2d.h:

```c
typedef struct
{
  int LattSize[2];
  int NumProcs[2];
  int Periods[2];
  int Reorder;
  int *Partition[2];
  int BoundaryWidth[2];
  size_t CellSize;
  MPI_Comm Communicator;
} GP2d_InputParams;
```
Figure 2: A typical configuration of the cellular automation after 1000 iterations started from a random distribution.

The array \texttt{LattSize} describes the lattice sizes of the lattice, the array \texttt{NumProcs} describes the lattice sizes of the “processor lattice”, i.e. the simulation is executed in parallel on a Cartesian lattice of processors of size \texttt{NumProcs[0]} × \texttt{NumProcs[1]}, the array \texttt{Periods} specifies whether the lattice is periodic (1) or not (0) in each dimension and the array \texttt{BoundaryWidth} describes the widths of the boundaries of the lattice that have to be communicated to the neighbouring processors. In general, these parameters are closely related to the “interaction radii” of the algorithms under consideration. For example, for an Ising model based on a nearest-neighbour interaction, the widths would be equal to one for all dimensions. The integer \texttt{Reorder} is used specify whether the MPI Library can rearrange(1) the processors while creating a virtual processor grid or not(0). The variable \texttt{ Communicator} contains an id that is used for communication in MPI. This id is given by a library initialization function. The array \texttt{Partition} consists of pointer to an int array of length \texttt{NumProcs[i]}. It defines the number of cells, which are distributed to the task in direction \texttt{i}. If the pointer is set to NULL the library will choose a balanced distribution. Fig. 3 shows a typical distribution of a lattice to 4 tasks with an interaction radius of one. The structure component \texttt{CellSize} finally defines the size of the “cell” (i.e. the size of the data structure associated with the lattice sites) itself, usually initialized by a call to the \texttt{sizeof} operator.

The struct \texttt{GP2d\_InputParams}, included in the struct \texttt{GP2d}, is then used by the routine \texttt{GP2d\_Initialize()} to initialize the generic data structure \texttt{GP2d} of \texttt{GeoPar} which contains the basic informations about the lattice and it’s distribution on the processors. For our example this would look like follows:

```c
GP2d\_InputParams InputParams;
GP2d *gp;
...
gp->InputParams.LattSize[0]=100;
```
Figure 3: Distributed array on 4 tasks. The dark grid cells are the "ghost" cells from the neighbored tasks.

```c
gp->InputParams.LattSize[1]=100;
gp->InputParams.NumProcs[0]=2;
gp->InputParams.NumProcs[1]=2;
gp->InputParams.BoundaryWidth[0]=1;
gp->InputParams.BoundaryWidth[1]=1;
gp->InputParams.Reorder=TRUE;
gp->InputParams.Periods[0]=TRUE;
gp->InputParams.Partition[0]=NULL;
gp->InputParams.CellSize=sizeof(STATE);
gp->InputParams.Communicator=MPI_COMM_WORLD;

MPI_Init(&argc, &argv);

GP2d_Initialize(gp);
```

Before the function `GP2d_Initialize()` is called, you must initialize the MPI environment by calling the function `MPI_Init(...)`. The function `GP2d_Initialize()` initializes the GP2d struct. All GeoPar functions need the pointer `gp` as their first argument, direct access to the GP2d struct should be avoided because future releases of GeoPar could change its structure.

The lattice itself is allocated by a call to `GP2d_CreateLatt()` returning a generic pointer of type `void **` that must be casted to the correct type. Fig. 4 illustrates the dynamic structure created by `GP2d_CreateLatt()`.

The content of the sublattices is undefined and may be initialized by using the `GP2d_FillLatt()` function, e.g.:

```c
STATE cell;
STATE **lattice;
cell=DEAD;
```
lattice=(STATE **)GP2d_CreateLatt(gp);
GP2d_FillLatt(gp,lattice,&cell);

Individual cells may be initialized by using GP2d_SetCell():

    cell=LIVE;
    GP2d_SetCell(gp,lattice,100,100,&cell);

The function is executed on every process, but only the process “owning” the cell updates its value and returns TRUE, all other tasks return FALSE.

2.4 Accessing the lattice

Direct access to the lattice is restricted to the local sublattice “owned” by the processor, including the “ghost elements” allocated to store data from neighbouring processes. The position of the local lattice (relative to the coordinates of the global lattice) can be determined by a call to the GP2d_GetLowCoordX1(), GP2d_GetHighCoordX1(), GP2d_LowCoordX2() and GP2d_GetHighCoordX2() functions. Direct access to a cell with coordinates \((x_1, x_2)\) is only possible if

\[
GP2d_{\text{GetLowCoordX1}}(gp) - GP2d_{\text{GetBoundaryWidth}}(gp)[0] \\
\leq x_1 < \\
GP2d_{\text{GetHighCoordX1}}(gp) + GP2d_{\text{GetBoundaryWidth}}(gp)[0]
\]

and

\[
GP2d_{\text{GetLowCoordX2}}(gp) - GP2d_{\text{GetBoundaryWidth}}(gp)[1] \\
\leq x_2 < \\
GP2d_{\text{GetHighCoordX2}}(gp) + GP2d_{\text{GetBoundaryWidth}}(gp)[1]
\]
holds.
The function \texttt{GP2d\_IsInMyLatt(gp,x1,x2)} returns a boolean value, whether
the coordinates are in the local lattice or not and can be used to test on the
validity of a read/write access.
Then the (preliminary) parallelized “update rule” for our example looks as
follows:

\begin{verbatim}
void updategol(GP2d *gp,STATE **newstate, const STATE **oldstate)
{
  int x1,x2;
  for(x1=GP2d\_GetLowCoordX1(gp);x1<GP2d\_GetHighCoordX1(gp);x1++)
    for(x2=GP2d\_GetLowCoordX2(gp);x2<GP2d\_GetHighCoordX2(gp);x2++)
      newstate[x1][x2]=updatestate(oldstate,x1,x2);
}
\end{verbatim}

This function updates in parallel the complete lattice. Because of iterations
over the complete lattice being performed frequently, \texttt{GeoPar} provides the
following macro \texttt{GP2d\_ForAll()} for performing the two nested loops:

\begin{verbatim}
GP2d\_ForAll(gp,x1,x2)
  newstate[x1][x2]=updatestate(oldstate,x1,x2);
\end{verbatim}

Accessing elements outside the boundaries of the local lattice results in an “ac-
cess violation” with unpredictable behavior. The \texttt{GP2d\_SetCell()} must be used
if the contents of a lattice cell has to be changed. function \texttt{GP2d\_IsInMyLatt()}
may be used to check whether the lattice element is local or not. To read the
contents of an arbitrary cell (local or not), the \texttt{GP2d\_GetCell()} function can
be used. The process owning the cell broadcasts it to all processes; therefore
calls of \texttt{GP2d\_GetCell()} should be restricted to a minimum.

\section{2.5 Updating the boundaries}
After initializing or updating the cells of the lattice, the “ghost cells” must
also be updated. This requires the communication of the boundary cells be-
tween processes. If we use periodic boundary conditions for the whole lattice,
\texttt{GP2d\_UpdBnd()} performs automatically the required communication processes.
Therefore, the final version of our update function looks as follows:

\begin{verbatim}
void updategol(GP2d *gp,STATE **newstate, const STATE **oldstate)
{
  int x1,x2;
  GP2d\_ForAll(gp,x1,x2)
    newstate[x1][x2]=updatestate(oldstate,x1,x2);
  GP2d\_UpdBnd(gp,newstate);
}
\end{verbatim}
2.6 I/O-Functions

The complete lattice can be read or written to a file by the use of `GP2d_FReadLatt()` respectively `GP2d_FWriteLatt()`. These functions provide a mechanism for compression of the data before writing them to a file by a call to a user defined function. This functions operates on one column of the local sublattice and writes the compressed data into a buffer. To compress the data from an int format to one byte one has to code

```c
void compresscolumn(GP2d *gp, char *buffer, const int *column)
{
    int x2;
    for(x2=GP2d_GetLowCoordX2(gp); x2<GP2d_GetHighCoordX2(gp); x2++)
        *buffer++ = (char)column[x2];
}
```

Using the above defined compress function, the configuration of our cellular automaton can be written to disk as follows:

```c
void writegol(GP2d *gp, STATE **lattice, FILE *file)
{
    GP2d_FWriteLatt(gp, lattice, file, compresscolumn, sizeof(char), TASK);
}
```

If no compression is necessary, instead of `compresscolumn()` a NULL pointer can be used in the call of `GP2d_FWriteLatt()`. Analogous to the `GP2d_FWriteLatt()` function it is possible to read the lattice using the `GP2d_FReadLatt()` function. Instead of a compress function a decompress function must be defined:

```c
void decompresscolumn(GP2d *gp, int *column, const char *buffer)
{
    int x2;
    for(x2=GP2d_GetLowCoordX2(gp); x2<GP2d_GetHighCoordX2(gp); x2++)
        column[x2] = (int)(*buffer++);
}
```

A complete listing of the parallelized “Game of Life” simulation using the GeoPar library is presented in the appendix.

2.7 Limitations of GeoPar

A 2-dimensional lattice of size $N \times M$ must consist of cell elements of the same type. The type of cell elements can be int, float or even complex structs with one restriction: All components must be static types, no pointers are allowed. For example
typedef int CELL1_T;
typedef float CELL2_T;
typedef struct
{
    int count;
    double koord[3];
}CELL3_T; /* Valid declarations */

are valid declarations, while

typedef struct
{
    int len;
    float *vector; /* Pointer to a vector */
}CELL_T; /* Invalid data type */

not. The interaction and dependencies between the cell elements must be local, i.e. limited to a finite distance. The efficiency is improved if this interaction radius is limited to the next neighborhood of the cells.

2.8 Performance considerations

In this section we want to estimate the efficiency, defined as

$$E(p) := \frac{T(1)}{p \times T(p)},$$

(1)

where $T(p)$ denotes the computing time of the algorithm on $p$ processors. If $T_{calc}$ is the time required for updating one cell, then the time of updating the whole lattice of $N$ cells is determined by

$$T(1) = N T_{calc}.\quad \text{(2)}$$

If the transfer of one cell element requires $T_{comm}$ time with a latency of $T_{lat}$, then the update of the neighbouring cells with interaction radius $k$ on $p$ processors requires

$$T_{update}(p) = 8 \left( T_{lat} + k \sqrt{\frac{N}{p} T_{comm}} \right). \quad \text{(3)}$$

Thus the computation time of the parallelized version on $p$ processors is

$$T(p) = \frac{N}{p} T_{calc} + 8 \left( T_{lat} + k \sqrt{\frac{N}{p} T_{comm}} \right). \quad \text{(4)}$$

Using eq. 2 and 4 we get for $E(p)$:

$$E(p) = \frac{1}{1 + 8 \left( \frac{p}{N} T_{calc} + \sqrt{\frac{k}{p} T_{comm}} \right)} \quad \text{(5)}$$

The efficiency is increased by increasing $N$. See, e.g. Fig. 5, which plots for different lattice sizes $N$ the efficiency as a function of $p$. 

14
Figure 5: Efficiency as a function of processor counts $p$ for different lattice sizes $N$. 

=800x800
=400x400
=200x200
=100x100
### 3 REFERENCE GUIDE

#### 3.1 Functions listed alphabetically

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP2d_BCastData</td>
<td>17</td>
<td>Broadcast data to each task</td>
</tr>
<tr>
<td>GP2d_Collect</td>
<td>19</td>
<td>Collect lattice to one task</td>
</tr>
<tr>
<td>GP2d_CopyLatt</td>
<td>23</td>
<td>Copy lattice</td>
</tr>
<tr>
<td>GP2d_CreateGlobalLatt</td>
<td>25</td>
<td>Create complete lattice</td>
</tr>
<tr>
<td>GP2d_CreateLatt</td>
<td>28</td>
<td>Create lattice</td>
</tr>
<tr>
<td>GP2d_FillLatt</td>
<td>30</td>
<td>Fill lattice with one cell</td>
</tr>
<tr>
<td>GP2d_FReadLatt</td>
<td>33</td>
<td>Read lattice from file</td>
</tr>
<tr>
<td>GP2d_Free</td>
<td>38</td>
<td>Free lattice structure</td>
</tr>
<tr>
<td>GP2d_FreeGlobalLatt</td>
<td>40</td>
<td>Free complete lattice</td>
</tr>
<tr>
<td>GP2d_FreeLatt</td>
<td>42</td>
<td>Free lattice</td>
</tr>
<tr>
<td>GP2d_FWriteLatt</td>
<td>44</td>
<td>Write lattice to file</td>
</tr>
<tr>
<td>GP2d_Gather</td>
<td>49</td>
<td>Gather lattice</td>
</tr>
<tr>
<td>GP2d_GetCell</td>
<td>52</td>
<td>Get cell</td>
</tr>
<tr>
<td>GP2d_Initialize</td>
<td>55</td>
<td>Initialization</td>
</tr>
<tr>
<td>GP2d_IsInInMyLatt</td>
<td>58</td>
<td>Test of membership</td>
</tr>
<tr>
<td>GP2d_ReduceData</td>
<td>61</td>
<td>Reduce data to local memory</td>
</tr>
<tr>
<td>GP2d_Scatter</td>
<td>65</td>
<td>Scatter lattice</td>
</tr>
<tr>
<td>GP2d_SetCell</td>
<td>68</td>
<td>Set cell</td>
</tr>
<tr>
<td>GP2d_UpdBnd</td>
<td>71</td>
<td>Update boundary with periodic boundary conditions</td>
</tr>
</tbody>
</table>
NAME  GP2d_BCastData - Broadcast data to each task. One task broadcast simple or complex data to the other tasks.

C SYNOPSIS

#include <GP2d.h>
void GP2d_BCastData(GP2d *gp, void *data,
                     size_t datasize, int source );

FORTRAN SYNOPSIS

include "GP2d.f"
GP2DF_BCastData(GP2d gp, INTEGER data,
                 INTEGER datasize, INTEGER source )

PARAMETERS

gp  The struct containing informational parameters.

data  Points to the data at the sender, or points to the the place where the
data can be stored at the receiver.

datasize  Size of the data in bytes.

source  The identifier of the sender.

DESCRIPTION  The aim of this function is to send a piece of memory to
all participating tasks. If you use this in Fortran 90, data will be the name of a
simple variable or struct. The task that id is given as source will be the sender
of the data, the other tasks will receive this data.

ERRORS

gp  Not initialized by GP2d_Init/GP2DF_Init.

source  Not in the range

- 0...GP2d_GetNumProcs(gp)[0]
   × GP2d_GetNumProcs(gp)[1] − 1, in C;
- 0...gp%NUMPROCS(1)
   ×gp%NUMPROCS(2) − 1, in Fortran 90.

C EXAMPLE

#include "GP2d.h"
#define ROOTTASK 0

void main(void)
{

GP2d gp = {{100,200},{4,4},{0,0},0,{NULL,NULL},
{1,1},sizeof(int),0,0},
0,0,0,0,{0,0},{0,0},{0,0},
{0,0},{0,0}};

GP2d *p_gp = &gp;
int dim[2] = {100, 200}, array[100][200], i, j;

GP2d_Initialize(p_gp);

if (GP2d_MyTaskId(p_gp) == ROOTTASK)
{
    for(i=0;i<dim[0];i++)
        for(j=0;j<dim[1];j++)
            array[i*dim[1]+j] = i*j;

    GP2d_BCastData(p_gp, array, sizeof(int)*dim[0]*dim[1],
                   ROOTTASK);
}

FORTRAN 90 EXAMPLE

include "GP2d.f"

program BCAST
use GEOPAR

    type (GP2D) GP
    integer ARRAY(0:99, 0:199), DIM(0:1)
    integer I, J, IERR

    DIM(0) = 100
    DIM(1) = 200
    GP%INPUTPARAMS%NUMPROCS(1) = GP2d_COMPUTE

    call MPI_INIT(IERR)
    call GP2DF_Initialize(GP, IERR)

    if (GP%MYTASKID == 0) then
        do I=0, DIM(0)-1
            do J=0, DIM(1)-1
                ARRAY(I,J) = I*J
            end do
        end do
    end if

    call GP2DF_BCastData(GP, ARRAY, 4*DIM(0)*DIM(1), 0)
end
NAME   GP2d_Collect - Collect lattice to one task. Simply collect the distributed lattice and put the sublattices at the correct (see GP2d_Gather?) places

C SYNOPSIS

#include <GP2d.h>
typedef void (*GP2d_CollectFunction) (GP2d*, void *,int, void *);
void GP2d_Collect(GP2d *gp, void *data, void **lattice,
                   GP2d_CollectFunction fcn, int taskid);

FORTRAN SYNOPSIS

include "GP2d.f"
GP2DF_Collect(GP2d gp, INTEGER data, INTEGER lattice,
              INTEGER fcn, INTEGER taskid)

PARAMETERS

gp   The pointer to the struct containing informational parameters.
data Some extra information that will be submitted to the “collect function” fcn.
lattice The pointer to the the distributed lattice.
fcn   Pointer to a function, that is executed on the process with taskid taskid.
      The first parameter of the function is a pointer to the sublattices, the
      second is the extra information (see parameter data), the third is the
      column number of the lattice, while the fourth is a pointer to one column
      of the whole lattice.
taskid The task identifier, where the data will be sent and the function will
        be evaluated.

DESCRIPTION   The aim of this function is to allow a column ordered output
              of the lattice to one designated task. The output function is defined by
              the third parameter and is called in column order for each column. This func-
              tion requires the sending of the whole lattice to one task, therefore excessive
              calls to this function should be avoided due to performance decrease. The data
              parameter can be used to submit information, like strings etc., to the collector.

ERRORS

gp   Not initialized by GP2d_Initialize/GP2DF_Initialize.
lattice Not allocated by GP2d_CreateLatt/GP2DF_CreateLatt.
fcn   Pointer to an invalid function.
taskid Not in the range

- $0 \ldots \text{GP2d\_GetNumProcs}(\text{gp})[0] \times \text{GP2d\_GetNumProcs}(\text{gp})[1] - 1$, in C;
- $0 \ldots \text{gp\%NUMPROCS}(1) \times \text{gp\%NUMPROCS}(2) - 1$, in Fortran 90.

C EXAMPLE

```c
#include <stdio.h>
#include <GP2d.h>
#define ROOTTASK 0

void printline(GP2d *gp,void *data, int x1,int *cell)
{
    int x2;
    printf("%3d ",x1);
    for(x2=0;x2<GP2d\_GetLattSize(gp)[1];x2++)
        if(cell[x2])
            printf("*");
        else
            printf(" ");
    printf("\n");
}

void initlattice(GP2d *gp,int **lattice)
{
    int x1,x2;
    GP2d\_ForAll(gp,x1,x2)
        lattice[x1][x2]=x1+x2;
}

void main(void)
{
    GP2d gp=\{\{40,10\}, \{2,2\}, \{True, True\}, \{True\},
        \{NULL,NULL\}, \{1,1\}, \{sizeof(int), 0, 0,
        0,0,0,0,0,0,0,\{0,0\},\{0,0\},\{0,0\},
        \{0,0\},\{0,0\},\{0,0\}\};
    GP2d *p_pg = &gp;
    int **lattice;
    GP2d\_Initialize(p_gp);
    lattice=(int **) GP2d\_CreateLatt(p_gp, NULL);
    initlattice(p_gp,lattice);
    GP2d\_Collect(p_gp,NULL,lattice,printline,ROOTTASK);
}
```

20
include "GP2d.f"

program COLLECT
use GEOPAR

subroutine INITLATT(LP, LATT)
use GEOPAR
type (GP2D) LP
integer LATT (LP%LOW(2):LP%HIGH(2), LP%LOW(1):LP%HIGH(1))

integer I, J

do I=LP%LOWCOORDS(1), LP%HIGHCOORDS(1)-1
  do J=LP%LOWCOORDS(2), LP%HIGHCOORDS(2)-1
    LATT(J,I) = J+I
  end do
end do
end subroutine INITLATT

type (GP2D) GP
integer, pointer :: ARRAY (:,:), IERR, PTR
external PrintColumn

GP%INPUTPARAMS%LATTSIZE(1) = 40
GP%INPUTPARAMS%LATTSIZE(2) = 20

GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

GP%INPUTPARAMS%NUMPROCS(1) = 2
GP%INPUTPARAMS%NUMPROCS(2) = 2

GP%INPUTPARAMS%REORDER = 1

GP%INPUTPARAMS%PERIODS(1) = 1
GP%INPUTPARAMS%PERIODS(2) = 1

GP%INPUTPARAMS%CELLSIZE = 4

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)

allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, PTR)
call INITLATT(GP, ARRAY)
call GP2DF_Collect(GP, 0, PTR, PrintColumn, 0)
end
NAME   GP2d_CopyLatt - Copy lattice.

C SYNOPSIS
#include <GP2d.h>
void GP2d_CopyLatt(GP2d *gp, void **dest, void **source);

FORTRAN SYNOPSIS
include "GP2d.f"
GP2DF_CopyLatt(GP2d gp, INTEGER dest, INTEGER source)

PARAMETERS

gp    Pointer to the struct containing informational parameters.

dest  Pointer to a sublattice to which the data should be copied.

source Pointer to the sublattice to be copied.

DESCRIPTION   The function copies the contents of one distributed lattice
to a second one.

ERRORS

gp    Not initialized by GP2d\_Initialize.

dest  Invalid pointer or not initialized by GP2d\_CreateLatt.

source Invalid pointer or not initialized by GP2d\_CreateLatt.

C EXAMPLE
#include <stdio.h>
#include <GP2d.h>
void main(void)
{
    GP2d gp = {{100,100},{2,2}, {True, True}, {True},
                {NULL,NULL}, {1,1}, sizeof(int), 0, 0,
                0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    GP2d *p_gp = &gp;
    int **lattice,**aux;
    GP2d\_Initialize(p_gp);

    lattice = (int **) GP2d\_CreateLatt(p_gp, NULL);
    aux     = (int **) GP2d\_CreateLatt(p_gp, NULL);
    GP2d\_CopyLatt(gp, aux, lattice);
}
FORTRAN 90 EXAMPLE

include "GP2d.f"

program COPY_LATT
use GEOPAR

  type (GP2D) GP
  integer, pointer :: SOURCE (:,:), DESTINATION (:,:)
  integer IERR, SOURCE_PTR, DESTINATION_PTR

  GP%INPUTPARAMS%LATTSIZE(1) = 100
  GP%INPUTPARAMS%LATTSIZE(2) = 100

  GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
  GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

  GP%INPUTPARAMS%NUMPROCS(1) = 2
  GP%INPUTPARAMS%NUMPROCS(2) = 2

  GP%INPUTPARAMS%REORDER = 1

  GP%INPUTPARAMS%PERIODS(1) = 1
  GP%INPUTPARAMS%PERIODS(2) = 1

  GP%INPUTPARAMS%CELLSIZE = 4

  call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)

allocate(SOURCE(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
allocate(DESTINATION(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, SOURCE, SOURCE_PTR)
call GP2DF_CreateLatt(GP, DESTINATION, DESTINATION_PTR)
call GP2DF_CopyLatt(GP, DESTINATION_PTR, SOURCE_PTR)
end
NAME   GP2d_CreateGlobalLatt - Create a complete lattice.

C SYNOPSIS

#include <GP2d.h>
void **GP2d_CreateGlobalLatt(GP2d *gp, void *lattice);

FORTRAN SYNOPSIS

include "GP2d.f"
GP2DF_CreateGlobalLatt(GP2d gp, INTEGER lattice, INTEGER ptr)

PARAMETERS

gp  Pointer to the struct containing informational parameters.

lattice In C equal to zero. Contains in Fortran a pointer to the array.

ptr  Contains a pointer to the lattice after the execution.

RETURN VALUE   A pointer to the new complete lattice.

DESCRIPTION   The function allocates memory for an undistributed lattice. One should be aware of the memory demands especially for large lattices and a large number of tasks.

ERRORS

gp  Not initialized by GP2d_Initialize/GP2DF_Initialize.

C EXAMPLE

#include <stdio.h>
#include <GP2d.h>
#define TASK 0
int main(void)
{
    GP2d gp = {{100,100},{2,2}, {True, True}, True,
               {NULL,NULL}, {1,1}, sizeof(int), 0, 0,
               0,0,0,0,0,0,0,0,0,0,0,
               {0,0},{0,0},{0,0}};
    GP2d *p_gp = &gp;
    int **lattice, **clattice;
    GP2d.Initialize(p_gp);
    lattice = (int **) GP2d_CreateLatt(p_gp);
    if(GP2d_GetMyTaskId(p_gp) == TASK)
    {
        clattice = (int **) GP2d_CreateGlobalLatt(p_gp);
    }
for(x1=0;x1<GP2d_GetLattSize(p_gp)[0];x1++)
    for(x2=0;x2<GP2d_GetLattSize(p_gp)[1];x2++)
        clattice[x1][x2] = 2;
}
GP2d_Scatter(gp, lattice, clattice, TASK);
}

FORTRAN 90 EXAMPLE

include "GP2d.f"

program CREATE_GLOBAL_LATT
use GEOPAR

type (GP2D) GP
integer, pointer :: ARRAY (:,:), GLOBAL_ARRAY (:,:)
integer IERR, ARRAY_PTR, GLOB_ARY_PTR, I, J

GP%INPUTPARAMS%LATTSIZE(1) = 100
GP%INPUTPARAMS%LATTSIZE(2) = 100

GP%INPUTPARAMS%NUMPROCS(1) = 2
GP%INPUTPARAMS%NUMPROCS(2) = 2

GP%INPUTPARAMS%PERIODS(1) = 1
GP%INPUTPARAMS%PERIODS(2) = 1

GP%INPUTPARAMS%REORDER = 1

GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

GP%INPUTPARAMS%CELLSIZE = 4

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)
allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)
if (GP%MYTASKID == 0) then
    allocate(GLOBAL_ARRAY(1:GP%INPUTPARAMS%LATTSIZE(2),
            1:GP%INPUTPARAMS%LATTSIZE(1)))
call GP2DF_CreateGlobalLatt(GP, GLOBAL_ARRAY, GLOB_ARY_PTR)
do I=1, GP%INPUTPARAMS%LATTSIZE(2)
do J=1, GP%INPUTPARAMS%LATTSIZE(1)
    GLOBAL_ARRAY(I,J) = 2
end do
end do
end if

call GP2DF_Scatter(GP, ARRAY_PTR, GLOB_ARY_PTR, 0)
end

RELATED INFORMATION

NAME   **GP2d_CreateLatt** - Create a distributed lattice.

C SYNOPSIS
#include <GP2d.h>
void **GP2d_CreateLatt(GP2d *gp, void *lattice);

FORTRAN SYNOPSIS
#include "GP2d.f"
GP2DF_CreateLatt(GP2d gp, INTEGER lattice, INTEGER ptr)

PARAMETERS
- **gp**  Pointer to the struct containing informational parameters.
- **lattice**  In C equal to zero. Contains in Fortran a pointer to the array.
- **ptr**  Contains a pointer to the lattice after the execution.

RETURN VALUE  A pointer to the new distributed lattice.

DESCRIPTION  The functions allocates a distributed lattice. The returned pointer should be casted to the type of the lattice cells.

ERRORS
- **gp**  Not initialized by **GP2d_Initilize/GP2DF_Initilize**.

EXAMPLES
#include <stdio.h>
#include <GP2d.h>
void main(void)
{
    GP2d gp = {{100,100},{2,2}, {True, True}, True,
               {NULL,NULL}, {1,1}, sizeof(int), 0, 0,
               0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    GP2d *p_gp = &gp;
    int **lattice;
    GP2d_Initialize(p_gp);
    lattice = (int **) GP2d_CreateLatt(p_gp);
}
FORTRAN 90 EXAMPLE

include "GP2d.f"

program CREATE_LATT
use GEOPAR

type (GP2D) GP
integer, pointer :: ARRAY (:,:)
integer IERR, ARRAY_PTR

GP%INPUTPARAMS%LATTSIZE(1) = 100
GP%INPUTPARAMS%LATTSIZE(2) = 100

GP%INPUTPARAMS%NUMPROCS(1) = 2
GP%INPUTPARAMS%NUMPROCS(2) = 2

GP%INPUTPARAMS%PERIODS(1) = 1
GP%INPUTPARAMS%PERIODS(2) = 1

GP%INPUTPARAMS%REORDER = 1

GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

GP%INPUTPARAMS%CELLSIZE = 4

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)

allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)
end

RELATED INFORMATION

Functions: GP2d_FreeLatt/GP2DF_FreeLatt.

29
NAME   GP2d_FillLatt - Fill lattice with one cell.

C SYNOPSI S
#include <GP2d.h>
void GP2d_FillLatt(GP2d *gp,void **lattice,const void *cell);

FORTRAN SYNOPSI S
include "GP2d.f"
GP2DF_FillLatt(GP2d gp, INTEGER lattice, INTEGER cell)

PARAMETERS
gp  The pointer to the struct containing informational parameters.

lattice  The pointer to the the distributed lattice.

cell  The lattice element that is copied to all cells of the lattice.

DESCRIPTION  The function fills the whole lattice with one lattice element. This routine is useful for initialization of the lattice configuration. The shadowed boundary cells are not filled and must be updated by a call to GP2d_UpdBnd/GP2DF_UpdBnd.

ERRORS

gp  Not initialized by GP2d_Initialize/GP2DF_Initialize.

lattice  Not allocated by GP2d_CreateLatt/GP2DF_CreateLatt.

cell  Not a valid pointer.

C EXAMPLE
#include <stdio.h>
#include <GP2d.h>
typedef struct
{
    int count;
    int value;
} CELL_T;

void main(void)
{
    CELL_T cell={10,213};
    GP2d gp = {{100,100},{2,2}, {True, True}, True,
               {NULL,NULL}, {1,1}, sizeof(CELL_T), 0, 0,
               0,0,0,0,0,{0,0},{0,0},{0,0},
               30
FORTRAN 90 EXAMPLE

include "GP2d.f"

module TYPES
   type CELL_T
      integer COUNT, VALUE
   end type CELL_T
end module TYPES

program FILL_LATT
use GEOPAR
use TYPES

type (GP2D) GP

type (CELL_T) CELL

type (CELL_T), pointer :: ARRAY (:,:), IERR, ARRAY_PTR

GP%INPUTPARAMS%LATTSIZE(1) = 100
GP%INPUTPARAMS%LATTSIZE(2) = 100

GP%INPUTPARAMS%NUMPROCS(1) = 2
GP%INPUTPARAMS%NUMPROCS(2) = 2

GP%INPUTPARAMS%PERIODS(1) = 1
GP%INPUTPARAMS%PERIODS(2) = 1

GP%INPUTPARAMS%REORDER = 1

GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

GP%INPUTPARAMS%CELLSIZE = 8

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)

allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)

CELL%COUNT = 10
CELL%VALUE = 213

end
NAME  GP2d_FReadLatt - Read lattice from a file. Simply read distributed lattice from file and distribute

C SYNONYMS

#include <stdio.h>
#include <GP2d.h>
int GP2d_FReadLatt(GP2d *gp,void **lattice,FILE *file,
                    GP2d_IOFunction iofcn,
                    GP2d_TransferFunction tfcn,
                    int bufelsize,int sourcetask);

FORTRAN SYNONYMS

include "GP2d.f"
GP2DF_FReadLatt(GP2d gp, INTEGER lattice, INTEGER file,
               INTEGER iofcn, INTEGER tfcn,
               INTEGER bufelsize, INTEGER sourcetask, INTEGER ierr)

PARAMETERS

gp  The pointer to the struct containing informational parameters.

lattice  The pointer to the distributed lattice.

file  The file descriptor the data is read. (You can open a file with the function
       GP2DF_FOpen in Fortran.)

iofcn  This function reads the data from the file. Here is a diagram of the
        principle of reading data from a file into a sublattice:

        file \rightarrow \text{iofcn} \rightarrow \text{tfcn} \rightarrow \text{sublattice}

        tfcn  A function implementing a transformation of the data after reading them
              from the file with iofcn. The function is called for all columns of
              the sublattice. If set to NULL, no decompression is performed.

bufelsize  The size of the data object stored in the file representing one cell.
           Is neglected, if iofcn is set to NULL.

sourcetask  The task identifier of the task, that reads the file and distributes
            the data to all other tasks.

 ierr  Return value.
DESCRIPTION  The function allows the restoring of a lattice configuration from one file. It is possible to decompress the data to save disk space and communication time. This decompress function(tfcn) processes one column of each sublattice. See the example for a correct implementation of a decompressing function. In GeoPar is one function GP2d_StdRead predefined, which performs no decompression at all. The boundary cells of the sublattices are not updated.

ERRORS

gp  Not initialized by GP2d_Initialize/GP2DF_Initialize.
lattice Not allocated by GP2d_CreateLatt/GP2DF_CreateLatt.
file  Invalid file pointer.
iofcn  Pointer to an invalid read function.
tfcn  Pointer to an invalid decompress function.
bufelsize  Less than zero.
sourcetask  Not in the range

- 0 ... GP2d_GetNumProcs(gp)[0] × GP2d_GetNumProcs(gp)[1] − 1, in C;
- 0...gp%NUMPROCS(1) ×gp%NUMPROCS(2) − 1, in Fortran 90.

C EXAMPLE
Program reads the contents of the lattice from a file.

```
#include <stdlib.h>
#include <stdio.h>
#include <GP2d.h>
#define ROOTTASK 0
typedef struct {
  int height;
  int color;
} surface;

void read_fcn(FILE *file, char *buffer, size_t *bufsize, int *nelem)
{
  int i;
  for(i=0;i<*nelem;i++)
  {
    fscanf(file, "(%i,%i)\n", buffer+i>(*bufsize),
    buffer+i>(*bufsize)+sizeof(int));
  }
```
void decompresscolumn(GP2d *gp, surface *column, char * buffer) {
    int x2;
    for(x2=GP2d_GetLowCoordX2(gp); x2<GP2d_GetHighCoordX2(gp); x2++) {
        column[x2].height = (int)(*buffer++);
        column[x2].color = (int)(*buffer++);
    }
}

int main(void) {
    GP2d gp = {{100,100},{2,2}, {True, True}, True,
               {NULL,NULL}, {1,1}, sizeof(surface), 0, 0,
               0,0,0,0,0,{0,0},{0,0},{0,0},
               {0,0},{0,0},{0,0}};
    GP2d *p_gp = &gp;
    surface **lattice;
    GP2d_Initialize(p_gp);
    lattice = (surface **) GP2d_CreateLatt(p_gp);
    FILE *file;
    if(GP2d_GetMyTaskId(p_gp)==ROOTTASK)
    if((file=fopen("test.cfg","rb"))==NULL) {
        fprintf(stderr,"Error opening file.
"");
        exit(EXIT_FAILURE);
    }
    GP2d_FReadLatt(p_gp, lattice, file, read_fcn, decompresscolumn,
                   sizeof(surface),ROOTTASK);
    if (GP2d_GetMyTaskId(p_gp)==ROOTTASK)
        fclose(file);
    GP2d_UpdBnd(p_gp, lattice);
    exit(EXIT_SUCCESS);
}

FORTRAN 90 EXAMPLE
There is a little difference to the C version. While in C it is assumed that the values in the file are readable, this is not so in FORTRAN.

include "GP2d.f"

module TYPES
    type SURFACE
        integer HEIGHT
integer COLOR
end type SURFACE
end module TYPES

subroutine READ_FCN(FILE, BUFFER, BUFSIZE, NELEM)
  use GEOPAR

  integer BUFSIZE, NELEM
  character, dimension(:) :: BUFFER
  integer FILE
  integer i,j

  do i=0, NELEM-1
    do j=1, BUFSIZE
      read (FILE, ADVANCE='NO') BUFFER(i*BUFSIZE+j)
    end do
  end do
end subroutine READ_FCN

subroutine DECOMPRESSCOLUMN(LP, COLUMN, BUFFER)
  use GEOPAR
  use TYPES

  type (GP2D) LP
  type (SURFACE) COLUMN(LP%LOWCOORDS(2):LP%HIGHCOORDS(2))
  integer BUFFER(0:2*(LP%HIGHCOORDS(2)-LP%LOWCOORDS(2)+1))
  integer i,j

  i=0
  do j=LP%LOWCOORDS(2), LP%HIGHCOORDS(2)-1
    COLUMN(j)%HEIGHT = BUFFER(i)
    i = i + 1
    COLUMN(j)%COLOR = BUFFER(i)
  end do
end subroutine DECOMPRESSCOLUMN

program FREAD_LATT
  use GEOPAR
  use TYPES

  type (GP2D) GP
  type (SURFACE), pointer :: ARRAY (:,:)
  integer IERR, ARRAY_PTR, FILE

  GP%INPUTPARAMS%LATTSIZE(1) = 100
  GP%INPUTPARAMS%LATTSIZE(2) = 100
GP%INPUTPARAMS%NUMPROCS(1) = 2
GP%INPUTPARAMS%NUMPROCS(2) = 2

GP%INPUTPARAMS%PERIODS(1) = 1
GP%INPUTPARAMS%PERIODS(2) = 1

GP%INPUTPARAMS%REORDER = 1

GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

GP%INPUTPARAMS%CELLSIZE = 8

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)

allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)

if (GP%MYTASKID == 0) then
  open(13, FILE='test.cfg', STATUS='OLD')
end if

call GP2DF_FReadLatt(GP, ARRAY_PTR, 13, READ_FCN,
  DECOMPRESSCOLUMN, 8, 0)
if (GP%MYTASKID == 0) then
  close(13)
end if

call GP2DF_UpdBnd(GP, ARRAY_PTR)
end

RELATED INFORMATION

**Functions:** GP2d_FStdReadLatt, GP2d_FWriteLatt/GP2DF_FWriteLatt.
NAME    GP2d_Free - Free lattice structure.

C SYNOPSIS

#include <GP2d.h>
void GP2d_Free(GP2d *gp);

FORTRAN SYNOPSIS

include "GP2d.f"
GP2DF_Free(GP2d gp)

PARAMETERS

gp    The pointer to the struct containing informational parameters.

DESCRIPTION This function deallocates the memory of the informational struct (in C) and releases the MPI-Communicator. After this operation no GeoPar routine except GP2d.Initialize/GP2DF.Initialize can be invoked.

ERRORS

gp    Not initialized by GP2d.Initialize/GP2DF.Initialize.

EXAMPLES

#include <stdio.h>
#include <GP2d.h>
void main(void)
{
    GP2d gp1 = {{100,100},{2,2}, {True, True}, True,
            {NULL,NULL}, {1,1}, sizeof(int), 0, 0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    GP2d *p_gp1 = &gp1;
    GP2d gp2 = {{50,50},{2,2}, {True, True}, True,
            {NULL,NULL}, {1,1}, sizeof(int), 0, 0,
            0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    GP2d *p_gp2 = &gp2;
    int **lattice;
    GP2d.Initialize(p_gp1);
    lattice = (int **) GP2d_CreateLatt(p_gp1);
    GP2d_Free(p_gp1);
    GP2d.Initialize(p_gp2);
}
FORTRAN 90 EXAMPLE

include "GP2d.f"

program FREE
use GEOPAR

  type (GP2D) GP
  integer, pointer :: ARRAY (:,:), IERR, ARRAY_PTR

  GP%INPUTPARAMS%LATTSIZE(1) = 100
  GP%INPUTPARAMS%LATTSIZE(2) = 100
  GP%INPUTPARAMS%NUMPROCS(1) = 2
  GP%INPUTPARAMS%NUMPROCS(2) = 2
  GP%INPUTPARAMS%PERIODS(1) = 1
  GP%INPUTPARAMS%PERIODS(2) = 1
  GP%INPUTPARAMS%REORDER = 1
  GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
  GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1
  GP%INPUTPARAMS%CELLSIZE = 8

  call MPI_INIT(IERR)
  call GP2DF_Initialize(GP, IERR)

  allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
  call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)

  call GP2DF_Free(GP)
  GP%INPUTPARAMS%LATTSIZE(1) = 50
  GP%INPUTPARAMS%LATTSIZE(2) = 50
  call GP2DF_Initialize(GP, IERR)
end

RELATED INFORMATION

Functions: GP2dInitialize/GP2DF_Initialize.
NAME  GP2_FreeGlobalLatt - Free complete lattice.

C SYNOPSIS

#include <GP2d.h>
void GP2d_FreeGlobalLatt(GP2d *gp, void **lattice);

FORTRAN SYNOPSIS

#include "GP2d.f"
GP2DF_FreeGlobalLatt(GP2d gp, INTEGER lattice)

PARAMETERS

gp  The pointer to the struct containing informational parameters.

lattice  Pointer to the complete lattice.

DESCRIPTION  This function deallocates the memory of a undistributed lattice.

ERRORS

gp  Not initialized by GP2d_Initialize/GP2DF_Initialize.

lattice  Not allocated by GP2d_CreateGlobalLatt/GP2DF_CreateGlobalLatt.

C EXAMPLE

#include <stdio.h>
#include <GP2d.h>
void main(void)
{
    GP2d gp = {{100,100},{2,2}, {True, True}, True,
               {NULL,NULL}, {1,1}, sizeof(int), 0, 0,
               0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    GP2d *p_gp = &gp;
    int **lattice;
    GP2d_Initialize(p_gp);
    lattice = (int **) GP2d_CreateGlobalLatt(p_gp);
    GP2d_FreeGlobalLatt(p_gp,lattice);
}
FORTRAN 90 EXAMPLE

include "GP2d.f"

program FREE_GLOBAL_LATT
use GEOPAR

  type (GP2D) GP
  integer, pointer :: ARRAY (:,:), IERR, ARRAY_PTR

  GP%INPUTPARAMS%LATTSIZE(1) = 100
  GP%INPUTPARAMS%LATTSIZE(2) = 100

  GP%INPUTPARAMS%NUMPROCS(1) = 2
  GP%INPUTPARAMS%NUMPROCS(2) = 2

  GP%INPUTPARAMS%PERIODS(1) = 1
  GP%INPUTPARAMS%PERIODS(2) = 1

  GP%INPUTPARAMS%REORDER = 1

  GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
  GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

  GP%INPUTPARAMS%CELLSIZE = 4

  call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)

  allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateGlobalLatt(GP, ARRAY, ARRAY_PTR)
call GP2DF_FreeGlobalLatt(GP, ARRAY_PTR)
end

RELATED INFORMATION

Functions: GP2d_CreateGlobalLatt/GP2DF_CreateGlobalLatt.
NAME GP2_FreeLatt - Free distributed lattice.

C SYNOPSIS
#include <GP2d.h>
void GP2d_FreeLatt(GP2d *gp, void **lattice);

FORTRAN SYNOPSIS
include "GP2d.f"
GP2DF_FreeLatt(GP2d gp, INTEGER lattice)

PARAMETERS

gp The pointer to the struct containing informational parameters.
lattice Pointer to the distributed lattice.

DESCRIPTION This function deallocates the memory of a distributed lattice.

ERRORS

gp Not initialized by GP2d_Initialize.
lattice Not allocated by GP2d_CreateLatt.

C EXAMPLE
#include <stdio.h>
#include <GP2d.h>
void main(void)
{
    GP2d gp = {{100,100},{2,2}, {True, True}, True,
               {NULL,NULL}, {1,1}, sizeof(int), 0, 0,
               0,0,0,0,0,{0,0},{0,0},{0,0},
               {0,0},{0,0},{0,0}};

    GP2d *p_gp = &gp;
    int **lattice;
    GP2d_Initialize(p_gp);
    lattice = (int **) GP2d_CreateLatt(p_gp);
    GP2d_FreeLatt(p_gp,lattice);
}
FORTRAN 90 EXAMPLE

include "GP2d.f"

program FREE_LATT
    use GEOPAR

    type (GP2D) GP
    integer, pointer :: ARRAY (:,:)
    integer IERR, ARRAY_PTR

    GP%INPUTPARAMS%LATTSIZE(1) = 100
    GP%INPUTPARAMS%LATTSIZE(2) = 100

    GP%INPUTPARAMS%NUMPROCS(1) = 2
    GP%INPUTPARAMS%NUMPROCS(2) = 2

    GP%INPUTPARAMS%PERIODS(1) = 1
    GP%INPUTPARAMS%PERIODS(2) = 1

    GP%INPUTPARAMS%REORDER = 1

    GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
    GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

    GP%INPUTPARAMS%CELLSIZE = 4

    call MPI_INIT(IERR)
    call GP2DF_Initialize(GP, IERR)

    allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
    call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)
    call GP2DF_FreeLatt(GP, ARRAY_PTR)
end

RELATED INFORMATION

Functions: GP2d_CreateLatt/GP2DF_CreateLatt.
NAME   GP2d_FWriteLatt - Write lattice to a file. Simply collect and write distributed lattice to file

C SYNOPSIS

#include <stdio.h>
#include <GP2d.h>
int GP2d_FWriteLatt(GP2d *gp, void* lattice, FILE *file, 
    GP2d_IOFunction iofcn, 
    GP2d_TransferFunction tfcn, 
    int bufelsize, int desttask);

FORTRAN SYNOPSIS

include "GP2d.f"
GP2DF_FWriteLatt(GP2d gp, INTEGER lattice, INTEGER file, 
    INTEGER iofcn, INTEGER tfcn, 
    INTEGER bufelsize, INTEGER desttask, INTEGER ierr)

PARAMETERS

gp  The pointer to the struct containing informational parameters.

lattice  The pointer to the distributed lattice.

file  The file descriptor the data is written.

iofcn  This function writes the data to the file. Here is a diagram of the principle of writing data from a sublattice into a file:

        sublattice  tfcn  iofcn  file

    tfcn  A function implementing a transformation of the data before writing them to the file with iofcn. The function is called for all columns of the sublattice. If set to NULL no compression is performed.

bufelsize  The size of the data object stored in the file representing one lattice element. Is neglected if iofcn is set to NULL.

desttask  The task identifier of the task, that collects the data and writes the file.

ierr  Return value.

DESCRIPTION  The function allows the storing of a lattice configuration to one file. It is possible via a definition of a function to compress the data to save disk space and communication time. This function processes one column of each sublattice. See the example for a correct implementation of a compressing function.
ERRORS

gp  Not initialized by GP2d_Initialize/GP2DF_Initialize.
lattice  Not allocated by GP2d_CreateLatt/GP2DF_CreateLatt.
file  Invalid file pointer.
iofcn  Pointer to an invalid write function.
tfcn  Pointer to an invalid compress function.
bufelsize  Less than zero.
desttask  Not in the range

  • 0 ... GP2d_GetNumProcs(gp)[0]
      × GP2d_GetNumProcs(gp)[1] − 1, in C;
  • 0 ... gp%NUMPROCS(1)
      ×gp%NUMPROCS(2) − 1, in Fortran 90.

C EXAMPLE
The program stores a surface lattice(see FReadLatt) reduced to two integer
into a file.

#include <stdlib.h>
#include <stdio.h>
#include <GP2d.h>
#define ROOTTASK 0
typedef struct {
    int height;
    int color;
} surface;

void write_fcn(FILE *file, char *buffer, size_t *bufsize, int *nelem) {
    int i;
    for(i=0;i<*nelem;i++) {
        fprintf(file, "(%i,%i)\n", buffer+i*(*bufsize),
                buffer+i*(*bufsize)+sizeof(int));
    }
}

void compresscolumn(GP2d *gp, char *buffer, surface *column) {
    int x2;
    for(x2=GP2d_GetLowCoordX2(gp);x2<GP2d_GetHighCoordX2(gp);x2++) {
    }
*buffer++ = (int)(column[x2].height);
*buffer++ = (int)(column[x2].color);

int main(void)
{
    GP2d gp = {{100,100},{2,2}, {True, True}, True,
                NULL,NULL}, {1,1}, sizeof(surface), 0, 0,
                0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                {0,0},{0,0},{0,0}};
    GP2d *p_gp = &gp;
    surface **lattice, cell = { 100, 22 }; 
    GP2d_Initialize(p_gp);
    lattice = (surface **) GP2d_CreateLatt(p_gp);
    FILE *file;

    GP2d_FillLatt(p_gp, lattice, &cell);
    if (GP2d_GetMyTaskId(p_gp) == ROOTTASK)
        if ((file=fopen("test.cfg","wb")) == NULL)
            {fprintf(stderr,"Error opening file.
              exit(EXIT_FAILURE);
    }

    GP2d_FWriteLatt(p_gp, lattice, file, write_fcn,
                    compresscolumn, sizeof(surface), ROOTTASK);
    if (GP2d_GetMyTaskId(p_gp) == ROOTTASK)
        fclose(file);
    exit(EXIT_SUCCESS);
}

FORTRAN 90 EXAMPLE
There is a little difference to the C version. While in C it is assumed that the
values in the file are readable, this is not so in FORTRAN.

include "GP2d.f"

module TYPES
    type SURFACE
        integer HEIGHT
        integer COLOR
    end type SURFACE
end module TYPES

subroutine WRITE_FCN(FILE, BUFFER, BUFSIZE, NELEM)
    use GEOPAR
integer BUFSIZE, NELEM
character, dimension(:) :: BUFFER
integer FILE
integer i,j

do i=0, NELEM-1
  do j=1, BUFSIZE
    write (FILE, ADVANCE='NO') BUFFER(i*BUFSIZE+j)
  end do
end do
end subroutine WRITE_FCN

subroutine COMPRESSCOLUMN(LP, BUFFER, COLUMN)
  use GEOPAR
  use TYPES
  type (GP2D) LP
  type (SURFACE) COLUMN(LP%LOWCOORDS(2):LP%HIGHCOORDS(2))
  integer BUFFER(0:2*(LP%HIGHCOORDS(2)-LP%LOWCOORDS(2)+1))
  integer i,j
  i=0
  do j=LP%LOWCOORDS(2), LP%HIGHCOORDS(2)-1
    BUFFER(i) = COLUMN(j)%HEIGHT
    i = i + 1
    BUFFER(i) = COLUMN(j)%COLOR
  end do
end subroutine COMPRESSCOLUMN

program FWRITE_LATT
  use GEOPAR
  use TYPES

  type (GP2D) GP
  type (SURFACE), pointer :: ARRAY (:,:)
  integer IERR, ARRAY_PTR, FILE
  type (SURFACE) CELL

  GP%INPUTPARAMS%LATTSIZE(1) = 100
  GP%INPUTPARAMS%LATTSIZE(2) = 100

  GP%INPUTPARAMS%NUMPROCS(1) = 2
  GP%INPUTPARAMS%NUMPROCS(2) = 2

  GP%INPUTPARAMS%PERIODS(1) = 1
  GP%INPUTPARAMS%PERIODS(2) = 1
GP%INPUTPARAMS%REORDER = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1
GP%INPUTPARAMS%CELLSIZE = 8

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)

allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)

if (GP%MYTASKID == 0) then
    open (15, FILE='test.cfg', STATUS='REPLACE')
end if

CELL%HEIGHT = 100
CELL%COLOR = 22
call GP2DF_FillLatt(GP, ARRAY_PTR, CELL)

call GP2DF_FWriteLatt(GP, ARRAY_PTR, 15, WRITE_FCN,
    COMPRESSCOLUMN, 8, 0)
if (GP%MYTASKID == 0) then
    close(15)
end if
end

RELATED INFORMATION

Functions: GP2d_FReadLatt/GP2DF_FReadLatt, GP2d_FStdWriteLatt.
NAME  GP2d_Gather - Gather lattice.

C SYNOPTIC

#include <GP2d.h>
void GP2d_Gather(GP2d *gp, void **clattice,
   const void **lattice, int taskid);

FORTRAN SYNOPTIC

include "GP2d.f"
GP2DF_Gather(GP2d gp, INTEGER clattice,
   INTEGER lattice, INTEGER taskid)

PARAMETERS

gp  The pointer to the struct containing informational parameters.

clattice  Pointer to complete lattice.

lattice  Pointer to distributed lattice.

taskid  task to which lattice is gathered.

DESCRIPTION  The function gathers a distributed lattice to one designated task. Only the task with id \textbf{task} must allocate storage for the complete lattice.

ERRORS

gp  Not initialized by GP2d_Initialize/GP2DF_Initialize.

clattice  Not allocated by GP2d_CreateGlobalLatt/GP2DF_CreateGlobalLatt in task \textbf{taskid}.

lattice  Not allocated by GP2d_CreateLatt/GP2DF_CreateLatt.

taskid  Not in the range

\begin{itemize}
  \item 0 \ldots GP2d_GetNumProcs(gp)[0] \\
    \times GP2d_GetNumProcs(gp)[1] - 1, in C;
  \item 0 \ldots gp\%NUMPROCS(1) \\
    \times gp\%NUMPROCS(2) - 1, in Fortran 90.
\end{itemize}
C EXAMPLE

```c
#include <stdio.h>
#include <GP2d.h>
#define TASK 0

void initlattice(GP2d *gp, int **lattice)
{
    int x1, x2;
    GP2d_ForAll(gp, x1, x2)
    lattice[x1][x2] = x1 * x2;
}

text int main(void)
{
    GP2d gp = {{100,100}, {2,2}, {True, True}, True,
               {NULL,NULL}, {1,1}, sizeof(int), 0, 0,
               0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    GP2d *p_gp = &gp;
    int **lattice, **clattice;
    FILE *file;
    int x1, x2;

    GP2d_Initialize(p_gp);
    lattice = (int **) GP2d_CreateLatt(p_gp);
    initlattice(p_gp, lattice);
    if (GP2d_GetMyTaskId(p_gp) == TASK)
        clattice = (int **) GP2d_CreateGlobalLatt(p_gp);
    GP2d_Gather(p_gp, clattice, lattice, TASK);
}
```

FORTRAN 90 EXAMPLE

```fortran
include "GP2d.f"

program GATHER
use GEOPAR

subroutine INITLATT(LP, LATT)
use GEOPAR
type (GP2D) LP
integer LATT (LP%LOW(2):LP%HIGH(2), LP%LOW(1):LP%HIGH(1))

integer I, J

do I=LP%LOWCOORDS(1), LP%HIGHCOORDS(1)-1
```
do J=LP%LOWCOORDS(2), LP%HIGHCOORDS(2)-1
    LATT(J,I) = J*I
end do
end do
end subroutine INITLATT

type (GP2D) GP
integer, pointer :: ARRAY (:,:), GLOBAL_ARRAY
integer IERR, ARRAY_PTR, GLOBAL_PTR

GP%INPUTPARAMS%LATTSIZE(1) = 100
GP%INPUTPARAMS%LATTSIZE(2) = 100

GP%INPUTPARAMS%NUMPROCS(1) = 2
GP%INPUTPARAMS%NUMPROCS(2) = 2

GP%INPUTPARAMS%PERIODS(1) = 1
GP%INPUTPARAMS%PERIODS(2) = 1

GP%INPUTPARAMS%REORDER = 1

GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

GP%INPUTPARAMS%CELLSIZE = 4

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)
allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)
call INITLATT(GP, ARRAY)
if (GP%MYTASKID == 0) then
    allocate(GLOBAL_ARRAY(1:GP%INPUTPARAMS%LATTSIZE(2),
                               1:GP%INPUTPARAMS%LATTSIZE(1)))
call GP2DF_CreateGlobalLatt(GP, GLOBAL_ARRAY, GLOBAL_PTR)
call GP2DF_Gather(GP, GLOBAL_PTR, ARRAY_PTR, 0)
end if
end

RELATED INFORMATION

Functions: GP2d_CreateGlobalLatt/GP2DF_CreateGlobalLatt, GP2d_Scatter/GP2DF_Scatter.
NAME   GP2d_GetCell - Get cell.

C SYNOPSIS

#include <GP2d.h>
int GP2d_GetCell(GP2d *gp, const void **lattice,
                 int x1, int x2, void *element);

FORTRAN SYNOPSIS

include "GP2d.f"
GP2DF_GetCell(GP2d gp, INTEGER lattice,
               INTEGER x1, INTEGER x2, INTEGER element)

PARAMETERS

gp  The pointer to the struct containing informational parameters.
lattice  The pointer to the the distributed lattice.
x1  x1-coordinate of the lattice element.
x2  x2-coordinate of the lattice element.
element  Pointer to a buffer, to which the contents of the lattice element is copied.

RETURN VALUE  TRUE, if the sublattice of the task contains the lattice element, FALSE, if not.

DESCRIPTION  This function allows an access to the contents of the whole lattice. The task containing the cell element broadcasts the data to all other tasks. Therefore the GP2d_SetCell/GP2DF_SetCell function is communication intensive, excessive calls to this function decreases performance significantly.

ERRORS

gp  Not initialized by GP2d_Initialize/GP2DF_Initialize.
lattice  Not allocated by GP2d_CreateLatt/GP2DF_CreateLatt.
x1  Not in the range
    • 0…GP2d_GetLattSize(gp)[0] − 1, in C;
    • 0…GP%INPUTPARAMS%LATTSIZE(1) − 1, in Fortran.
x2  Not in the range
    • 0…GP2d_GetLattSize(gp)[1] − 1, in C;
    • 0…GP%INPUTPARAMS%LATTSIZE(2) − 1, in Fortran.
C EXAMPLE

```c
#include <stdio.h>
#include <GP2d.h>

void initlattice(GP2d *gp, int **lattice)
{
    int x1, x2;
    GP2d_ForAll(gp, x1, x2)
        lattice[x1][x2] = x1 + x2;
}

void main(void)
{
    GP2d gp = {{100, 100}, {2, 2}, {true, true}, true,
               {NULL, NULL}, {1, 1}, sizeof(int), 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    GP2d *p_gp = &gp;
    int element;
    GP2d.Initialize(p_gp);
    lattice = (int **) GP2d_CreateLatt(p_gp);
    initlattice(p_gp, lattice);
    GP2d_GetCell(p_gp, lattice, 50, 50, &element);
}
```

FORTRAN 90 EXAMPLE

```fortran
include "GP2d.f"

program GET_CELL
use GEOPAR

subroutine INITLATTICE(LP, LATT)
    use GEOPAR
type (GP2D) LP
    integer LATT (LP%LOW(2):LP%HIGH(2), LP%LOW(1):LP%HIGH(1))

    integer I, J
    do I = LP%LOWCOORDS(1), LP%HIGHCOORDS(1)-1
        do J = LP%LOWCOORDS(2), LP%HIGHCOORDS(2)-1
            LATT(J, I) = J*I
        end do
    end do
end subroutine INITLATTICE
```
type (GP2D) GP
integer, pointer :: ARRAY (:,:), IERR, ARRAY_PTR, ELEMENT

GP%INPUTPARAMS%LATTSIZE(1) = 100
GP%INPUTPARAMS%LATTSIZE(2) = 100

GP%INPUTPARAMS%NUMPROCS(1) = 2
GP%INPUTPARAMS%NUMPROCS(2) = 2

GP%INPUTPARAMS%PERIODS(1) = 1
GP%INPUTPARAMS%PERIODS(2) = 1

GP%INPUTPARAMS%REORDER = 1

GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

GP%INPUTPARAMS%CELLSIZE = 4

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)
allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)
call INITLATTICE(GP, ARRAY)
call GP2DF_GetCell(GP, ARRAY_PTR, 50, 50 ELEMENT)
end

RELATED INFORMATION

NAME    GP2d_Initialize - Initialization.

C SYNOPSIS

#include <GP2d.h>
int GP2d_Initialize(GP2d *gp);

PARAMETERS

gp  Pointer to the struct defining the lattice size, etc.
ret  Return value.

RETURN VALUE    Returns whether the execution was successful or not.

STRUCTURE

typedef struct
{
    int LattSize[2];
    int NumProcs[2];
    int Periods[2];
    int Reorder;
    int *Partition[2];
    int BoundaryWidth[2];
    size_t CellSize;
} GP2d_InputParams;

Description of the components:

LattSize  Denotes the lattice size in $x_1, x_2$ direction.

NumProcs  Defines the size of the task matrix. The lattice is distributed on
          a $\text{NumProcs}[0] \times \text{NumProcs}[1]$ matrix.

Periods  If one direction is set true(1), the first and the last cell of the lattice
          in this direction are neighbours.

Reorder  Allows the MPI library to rearrange the underlying processors grid.

Partition  Defines the partition of the distributed lattice over the task matrix.
         If set to NULL, the library will set a balanced distribution.

BoundaryWidth  Defines the range of interaction for each $x_i$-direction. BoundaryWidth[$i$]
               cells are shadowed in each task and updated by via a call to GP2dUpdBnd/GP2DFUpdBnd.

CellSize  Defines the cell size of the lattice.
DESCRIPTION  The function initializes the informational structures necessary for the GeoPar library. Every GeoPar function uses the pointer gp as its first argument. If the components of the vector NumProcs are set to GP2d_COMPUTE, the size of the task matrix will be computed from the number of tasks available.

C EXAMPLE  The program defines an int $100 \times 100$ lattice distributed on a $2 \times 4$ task matrix.

```c
#include <stdlib.h>
#include <stdio.h>
#include <GP2d.h>

int main(void)
{
    GP2d gp;
    GP2d *p_gp = &gp;

    int *partvec[4]={10,40,40,10};
    gp.InputParams.LattSize[0]=100;
    gp.InputParams.LattSize[1]=100;
    gp.InputParams.NumProcs[0]=2;
    gp.InputParams.Periods[0]=1;
    gp.InputParams.Reorder=0;
    gp.InputParams.Partition[0]=NULL;
    gp.InputParams.BoundaryWidth[0]=1;
    gp.InputParams.CellSize=sizeof(int);

    MPI_Init(NULL,NULL);
    if(!GP2d_Initialize(p_gp))
    {
        fprintf(stderr,"Error initializing lattice.\n");
        exit(EXIT_FAILURE);
    }
    exit(EXIT_SUCCESS);
}
```

FORTRAN 90 EXAMPLE

```fortran
include "GP2d.f"

program INITIALIZE
use GEOPAR
```

56
type (GP2D) GP
integer IERR

GP%INPUTPARAMS%LATTSIZE(1)=100;
GP%INPUTPARAMS%LATTSIZE(2)=100;
GP%INPUTPARAMS%NUMPROCS(1)=2;
GP%INPUTPARAMS%NUMPROCS(2)=4;
GP%INPUTPARAMS%PERIODS(1)=1;
GP%INPUTPARAMS%PERIODS(2)=1;
GP%INPUTPARAMS%REORDER=0;
GP%INPUTPARAMS%PARTITION(1)=0;
GP%INPUTPARAMS%PARTITION(2)=0;
GP%INPUTPARAMS%BOUNDARYWIDTH(1)=1;
GP%INPUTPARAMS%BOUNDARYWIDTH(2)=1;
GP%INPUTPARAMS%CELLSIZE=4;

call MPI_INIT(IERR);
call GP2DF_Initialize(GP, IERR)
}

RELATED INFORMATION

Functions: GP2d_Free/GP2DF_Free
NAME   GP2d_IsinMyLatt - Test on membership.

C SYNOPSIS
#include <GP2d.h>
int GP2d_IsinMyLatt(GP2d *gp, int x1, int x2);

FORTRAN SYNOPSIS
#include "GP2d.f"
GP2DF_IsinMyLatt(GP2D gp, INTEGER x1,
                INTEGER x2, INTEGER ret)

PARAMETERS
gp  The pointer to the struct containing informational parameters.

x1  The x1-coordinate.
x2  The x2-coordinate.
ret Return value.

RETURN VALUE  TRUE, if element (x1, x2) is in the local sublattice of the
task, FALSE otherwise.

DESCRIPTION  Functions tests if element (x1, x2) is member of the local
sublattice.

ERRORS
gp  Not initialized by GP2d.Initialize/GP2DF.Initialize.

C EXAMPLE  The program initializes a lattice to 1, the cell at the center
is increase by 1.
#include <stdlib.h>
#include <stdio.h>
#include <GP2d.h>
#define sizex 100
#define sizey 100

int main(void)
{
  GP2d gp = {{sizex, sizey}, {2, 2}, {True, True}, True,
              {NULL, NULL}, {1, 1}, sizeof(int), 0, 0,
              0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
  GP2d *p_gp = &gp;
#include "GP2d.f"

program ISIN_MY_LATT
use GEOPAR

type (GP2D) GP
integer, pointer :: ARRAY (:,:), ELEMENT
integer IERR, ARRAY_PTR

GP%INPUTPARAMS%LATTSIZE(1) = 100
GP%INPUTPARAMS%LATTSIZE(2) = 100

GP%INPUTPARAMS%NUMPROCS(1) = 2
GP%INPUTPARAMS%NUMPROCS(2) = 2

GP%INPUTPARAMS%PERIODS(1) = 1
GP%INPUTPARAMS%PERIODS(2) = 1

GP%INPUTPARAMS%REORDER = 1

GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

GP%INPUTPARAMS%CELLSIZE = 4

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)
allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)
ELEMENT = 1
call GP2DF_FillLatt(GP, ARRAY_PTR, ELEMENT)
call GP2DF_IsinMyLatt(GP, 50, 50, IERR)
if (IERR == 1) then
ARRAY(50,50) = ARRAY(50,50) + 1
end if
end

RELATED INFORMATION

NAME GP2d_ReduceData - Reduce distributed data to local data.

C SYNOPSIS

#include <GP2d.h>
void GP2d_ReduceData(GP2d *gp, void *OutMsg, void *InMsg,
        size_t Length, MPI_Datatype Datatype,
        int Task, MPI_Op Operation);

FORTRAN SYNOPSIS

include "GP2d.f"
GP2DF_ReduceData(GP2D gp, INTEGER OutMsg, INTEGER InMsg,
        INTEGER Length, INTEGER Datatype,
        INTEGER Task, INTEGER Operation)

PARAMETERS

gp Struct that contains informational parameters.
OutMsg Pointer to the source data.
InMsg Pointer to the result data (i.e. local data of one task).
Length The number of items in the result.
Datatype Type of the operands (to which OutMsg points).
Task The number of the task that gets the result.
Operation The operation that will be performed. Look into your MPI-manual.

DESCRIPTION The function submits the parameters to the corresponding
MPI function, which performs a reduce operation on all values to a single value.
You may have a look into the reference manual of MPI for information about
predefined reduction operations and valid datatypes (IBM Parallel Environment
for AIX: MPI Programming and Subroutine Reference, 1995).

ERRORS

gp Not initialized with GP2d.Initialize/GP2DF.Initialize.
OutMsg, InMsg Invalid pointers.
Datatype Unknown datatype.
Task Not in the range

• 0 ... GP2d_GetNumProcs(gp)[0]
  × GP2d_GetNumProcs(gp)[1] − 1, in C;
0...gp%NUMPROCS(1)
\times gp%NUMPROCS(2) - 1, in Fortran 90.

**Operation** Unknown operation.

**C EXAMPLE**

```c
#include <GP2d.h>
#include <stdlib.h>
#include <time.h>

#define TASK 0

int main(void)
{
    GP2d gp = {{100,100},{2,2}, {True, True}, True,
               {NULL,NULL}, {1,1}, sizeof(int), 0, 0,
               0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
    GP2d *p_gp = &gp;
    int **lattice;
    int x1, x2, sum, count;

    GP2d_Initialize(p_gp);
    lattice = (int **) GP2d_CreateLatt(p_gp);

    seed((int)time(NULL));
    GP2d_ForAll(p_gp,x1,x2)
    {
        lattice[x1][x2] = random() % 2;
    }

    count = 0;
    GP2d_ForAll(p_gp,x1,x2)
    {
        count += lattice[x1][x2];
    }

    GP2d_ReduceData(p_gp, &count, &sum, 1, MPI_INT, TASK, MPI_SUM);
    if (GP2d_GetMyTaskId(p_gp) == TASK)
        printf("number of initialized cells: %i\n", sum);
}
```

**FORTRAN 90 EXAMPLE**

```fortran
include "GP2d.f"
```

62
program REDUCEDATA
use GEOPAR

type (GP2D) GP
integer, pointer :: ARRAY (:,:)
integer IERR, ARRAY_PTR, X1, X2, count, sum
real, intent(out) :: value

GP%INPUTPARAMS%LATTSIZE(1) = 100
GP%INPUTPARAMS%LATTSIZE(2) = 100

GP%INPUTPARAMS%NUMPROCS(1) = 2
GP%INPUTPARAMS%NUMPROCS(2) = 2

GP%INPUTPARAMS%PERIODS(1) = 1
GP%INPUTPARAMS%PERIODS(2) = 1

GP%INPUTPARAMS%REORDER = 1

GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

GP%INPUTPARAMS%CELLSIZE = 4

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)

allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)

do X1=GP%LOWCOORDS(1), GP%HIGHCOORDS(1)-1
   do X2=GP%LOWCOORDS(2), GP%HIGHCOORDS(2)-1
      call RANDOM_NUMBER(value)
      if (value >= 0.5) then ARRAY(X2,X1)=1 endif
   end do
end do

count = 0
do X1=GP%LOWCOORDS(1), GP%HIGHCOORDS(1)-1
   do X2=GP%LOWCOORDS(2), GP%HIGHCOORDS(2)-1
      count = count + ARRAY(X2,X1)
   end do
end do

call GP2DF_ReduceData(GP, count, sum, 1, MPI_INTEGER, 0, MPI_SUM)
if (GP%MYTASKID == 0) then
    print *, 'number of initialized cells: ',sum
end if
end
NAME   GP2d_Scatter - Scatter lattice.

C SYNONYMS
#include <GP2d.h>
void GP2d_Scatter(GP2d *gp,void **lattice,const void **clattice,int task);

FORTRAN SYNONYMS
include "GP2d.f"
GP2DF_Scatter(GP2D gp, INTEGER lattice, INTEGER clattice, INTEGER task)

PARAMETERS
gp    The pointer to the struct containing informational parameters.
lattice Pointer to distributed lattice(own sublattice).
clattice Pointer to complete lattice(only for task necessary).
task  Task is the owner of the complete lattice.

DESCRIPTION    The function scatters a complete lattice from one task to
a distributed lattice(each task owns a sublattice).

ERRORS
gp    Not initialized by GP2d_Initialize/GP2DF_Initialize.
lattice Not allocated by GP2d_CreateLatt/GP2DF_CreateLatt.
clattice Not allocated by GP2d_CreateGlobalLatt/GP2DF_CreateGlobalLatt
        in task task.
task  Not in the range
        • 0 ... GP2d_GetNumProcs(gp)[0]
          × GP2d_GetNumProcs(gp)[1] − 1, in C;
        • 0...gp%NUMPROCS(1)
          ×gp%NUMPROCS(2) − 1, in Fortran 90.

C EXAMPLE
#include <GP2d.h>
#define TASK 0

int main(void)
{
    GP2d gp = {{100,100},{2,2}, {True, True}, True,
GP2d *p_gp = &gp;
int **lattice, **clattice;
int x1, x2;

GP2d_Initialize(p_gp);
lattice = (int **) GP2d_CreateLatt(p_gp);

if(GP2d_GetMyTaskId(p_gp) == TASK)
{
    clattice=(int **)GP2d_CreateGlobalLatt(p_gp);
    for(x1=0;x1<GP2d_GetLattSize(p_gp)[0];x1++)
        for(x2=0;x2<GP2d_GetLattSize(p_gp)[1];x2++)
            *(clattice[x1]+x2) = x1 + x2;
}

GP2d_Scatter(p_gp, lattice, clattice, TASK);
if(GP2d_GetMyTaskId(p_gp)==TASK)
    GP2d_FreeGlobalLatt(p_gp, clattice);
    GP2d_UpdBnd(gp, lattice);
}

FORTRAN 90 EXAMPLE

include "GP2d.f"

program SCATTER
use GEOPAR

type (GP2D) GP
integer, pointer :: ARRAY (:,:), CARRAY (:,:)
integer IERR, ARRAY_PTR, C_PTR, X1, X2

GP%INPUTPARAMS%LATTSIZE(1) = 100
GP%INPUTPARAMS%LATTSIZE(2) = 100

GP%INPUTPARAMS%NUMPROCS(1) = 2
GP%INPUTPARAMS%NUMPROCS(2) = 2

GP%INPUTPARAMS%PERIODS(1) = 1
GP%INPUTPARAMS%PERIODS(2) = 1

GP%INPUTPARAMS%REORDER = 1

GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

GP%INPUTPARAMS%CELLSIZE = 4

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)

allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)

if (GP%MYTASKID == 0) then
allocate(CARRAY(1:GP%INPUTPARAMS%LATTSIZE(2),
1:GP%INPUTPARAMS%LATTSIZE(1)))
call GP2DF_CreateGlobalLatt(GP, CARRAY, C_PTR)
do X1=1, GP%INPUTPARAMS%LATTSIZE(2)
do X2=1, GP%INPUTPARAMS%LATTSIZE(1)
   CARRAY(X1,X2) = X1 + X2
end do
end do
end if

GP2DF_Scatter(GP, ARRAY_PTR, C_PTR, 0)

if (GP%MYTASKID == 0) then
   call GP2DF_FreeGlobalLatt(C_PTR)
end if

call GP2DF_UpdBnd(GP, ARRAY_PTR)
end

RELATED INFORMATION

Functions: GP2d_CreateCLatt, GP2d_Gather.
NAME GP2d_SetCell - Set cell.

C SYNOPSIS
#include <GP2d.h>
int GP2d_SetCell(GP2d *gp, void **lattice,
                 int x1, int x2, void *cell);

FORTRAN SYNOPSIS
include "GP2d.f"
GP2DF_SetCell(GP2D gp, INTEGER lattice,
             INTEGER x1, INTEGER x2, INTEGER cell,
             INTEGER ret)

PARAMETERS
  gp  The pointer to the struct containing informational parameters.
  lattice The pointer to the the distributed lattice.
  x1  The $x_1$-coordinate.
  x2  The $x_2$-coordinate.
  cell A pointer to the new item written at $(x_1, x_2)$.
  ret Return value.

RETURN VALUE  TRUE, if element $(x_1, x_2)$ is in the local sublattice of the task, FALSE otherwise.

DESCRIPTION Functions writes a new element into the lattice and is performed on all tasks. Only the task containing the new lattice element is updated. The shadowed boundary elements are not updated.

ERRORS
  gp  Not initialized by GP2d_Initialize/GP2DF_Initialize.
  lattice Not allocated by GP2d_CreateLatt/GP2DF_CreateLatt.
  x1  Not in the range
       • 0...GP2d_GetLattSize(gp)[0] − 1, in C;
       • 0...gp%INPUTPARAMS%LATTSIZE(1) − 1, in Fortran.
  x2  Not in the range
       • 0...GP2d_GetLattSize(gp)[1] − 1, in C;
       • 0...gp%INPUTPARAMS%LATTSIZE(2) − 1, in Fortran.
  cell Not a valid pointer.
C EXAMPLE  The program sets the center of the lattice to 1, all other elements are initialized to zero.

```c
#include <stdlib.h>
#include <stdio.h>
#include <GP2d.h>
#define sizex 100
#define sizey 100

int main(void)
{
    GP2d gp = {{sizex, sizey}, {2, 2}, {True, True}, True,
               {NULL, NULL}, {1, 1}, sizeof(int), 0, 0,
               0, 0, 0, 0, {0, 0}, {0, 0}, {0, 0},
               {0, 0}, {0, 0}, {0, 0}};
    GP2d *p_gp = &gp;
    int **lattice;
    int element;
    GP2d_Initialize(p_gp);
    lattice = (int **) GP2d_CreateLatt(p_gp);
    element = 0;
    GP2d_FillLatt(p_gp, lattice, &element);
    GP2d_SetCell(p_gp, lattice, sizex/2, sizey/2, &element);
}
```

FORTRAN 90 EXAMPLE

```fortran
include "GP2d.f"

program SET_CELL
use GEOPAR

type (GP2D) GP
    integer, pointer :: ARRAY (:,:)    
    integer IERR, ARRAY_PTR, ELEMENT

    GP%INPUTPARAMS%LATTSIZE(1) = 100
    GP%INPUTPARAMS%LATTSIZE(2) = 100

    GP%INPUTPARAMS%NUMPROCS(1) = 2
    GP%INPUTPARAMS%NUMPROCS(2) = 2

    GP%INPUTPARAMS%PERIODS(1) = 1
    GP%INPUTPARAMS%PERIODS(2) = 1

    GP%INPUTPARAMS%REORDER = 1
```
GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

GP%INPUTPARAMS%CELLSIZE = 4

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)

allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, ARRAY_PTR)
ELEMENT = 0
call GP2DF_FillLatt(GP, ARRAY_PTR, ELEMENT)
call GP2DF_SetCell(GP, 50, 50, ELEMENT, IERR)
end

RELATED INFORMATION

NAME  GP2d_UpdBnd - Update boundaries with periodic boundary conditions.

C SYNOPSIS
#include <GP2d.h>
void GP2d_UpdBnd(GP2d *gp, void **lattice);

FORTRAN SYNOPSIS
include "GP2d.f"
GP2DF_UpdBnd(GP2D gp, INTEGER lattice)

PARAMETERS

  gp  The pointer to the struct containing informational parameters.

  lattice The pointer to the the distributed lattice.

DESCRIPTION  After an update of the lattice the shadowed boundary cells must also be updated. Boundary cells must be interchanged between adjacent tasks. The function performs this update process.

ERRORS

  gp  Not initialized by GP2d_Initialize/GP2DF_Initialize.

  lattice Not allocated by GP2d_CreateLatt/GP2DF_CreateLatt.

C EXAMPLE

#include <stdlib.h>
#include <stdio.h>
#include <GP2d.h>
#define n 100 /* number of iterations */

void initlattice(GP2d *gp, int **lattice)
{
    int cell;
    cell=0;
    GP2d_FillLatt(gp, lattice, &cell);
    cell=1;
    GP2d_SetCell(gp, lattice, GP2d_GetLattSize(gp)[0]/2, 
                   GP2d_GetLattSize(gp)[1]/2, &cell);
    GP2d_UpdBnd(gp, lattice);
}

void updatelattice(GP2d *gp, int **lattice)
{
int x1, x2;
GP2d_ForAll(gp, x1, x2)
    lattice[x1][x2] = (lattice[x1+1][x2]+lattice[x1-1][x2]+
                    lattice[x1][x2+1]+lattice[x1][x2-1])/4;
GP2d_UpdBnd(gp, lattice);
}

int main(void)
{
    GP2d gp = {{100, 100}, {2, 2}, {True, True}, True,
               {NULL, NULL}, {1, 1}, sizeof(int), 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0};
    GP2d *p_gp = &gp;
    int **lattice;
    int i;
    GP2d.Initialize(p_gp);
    lattice = (int **) GP2d_CreateLatt(p_gp);

    initlattice(gp, lattice);
    for (i=0; i<n; i++)
        updatelattice(gp, lattice);
}

FORTRAN 90 EXAMPLE

include "GP2d.f"

program UPDBND
use GEOPAR

subroutine INIT_LATTICE(LP, LATT_PTR)
    use GEOPAR
    type (GP2D) LP

    integer IERR, CELL

    CELL = 0
    call GP2DF_FillLatt(LP, LATT_PTR, CELL)
    CELL = 1
    call GP2DF_SetCell(LP, LATT_PTR, 50, 50, CELL, IERR)
    call GP2DF_UpdBnd(LP, LATT_PTR)
end subroutine INIT_LATTICE

subroutine UPDATE_LATTICE(LP, LATT, LATT_PTR)
    use GEOPAR
    type (GP2D) LP

    72
integer LATT (LP%LOW(2):LP%HIGH(2), LP%LOW(1):LP%HIGH(1))

integer I, J

do I=LP%LOWCOORDS(1), LP%HIGHCOORDS(1)-1
    do J=LP%LOWCOORDS(2), LP%HIGHCOORDS(2)-1
        LATT(J,I) = (LATT(J-1,I) + LATT(J+1,I) + LATT(J,I-1) + LATT(J,I+1))/4
    end do
end do
call GP2DF_UpdBnd(LP, LATT_PTR)
end subroutine UPDATE_LATTICE

type (GP2D) GP
integer, pointer :: ARRAY (:,:)
integer IERR, ARRAY_PTR

GP%INPUTPARAMS%LATTSIZE(1) = 100
GP%INPUTPARAMS%LATTSIZE(2) = 100

GP%INPUTPARAMS%BOUNDARYWIDTH(1) = 1
GP%INPUTPARAMS%BOUNDARYWIDTH(2) = 1

GP%INPUTPARAMS%NUMPROCS(1) = 2
GP%INPUTPARAMS%NUMPROCS(2) = 2

GP%INPUTPARAMS%REORDER = 1

GP%INPUTPARAMS%PERIODS(1) = 1
GP%INPUTPARAMS%PERIODS(2) = 1

GP%INPUTPARAMS%CELLSIZE = 4

call MPI_INIT(IERR)
call GP2DF_Initialize(GP, IERR)

allocate(ARRAY(GP%LOW(2):GP%HIGH(2), GP%LOW(1):GP%HIGH(1)))
call GP2DF_CreateLatt(GP, ARRAY, PTR)
call INIT_LATTICE(GP, ARRAY_PTR)
do I=1, 100
    call UPDATE_LATTICE(GP, ARRAY, ARRAY_PTR)
end do
### 3.2 Macros listed alphabetically

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP2d_ForAll</td>
<td>75</td>
<td>Iterator over whole sublattice</td>
</tr>
<tr>
<td>GP2d_FStdReadLatt</td>
<td>76</td>
<td>Read lattice from file</td>
</tr>
<tr>
<td>GP2d_FStdWriteLatt</td>
<td>78</td>
<td>Write lattice to file</td>
</tr>
<tr>
<td>GP2d_GetBoundaryWidth</td>
<td>80</td>
<td>Get width of boundaries</td>
</tr>
<tr>
<td>GP2d_GetCellSize</td>
<td>81</td>
<td>Get size of cell element</td>
</tr>
<tr>
<td>GP2d_GetHighCoordX1</td>
<td>82</td>
<td>Get upper boundary of sublattice</td>
</tr>
<tr>
<td>GP2d_GetHighCoordX2</td>
<td>83</td>
<td>Get upper boundary of sublattice</td>
</tr>
<tr>
<td>GP2d_GetLattSize</td>
<td>84</td>
<td>Get size of global lattice</td>
</tr>
<tr>
<td>GP2d_GetLowCoordX1</td>
<td>85</td>
<td>Get lower bound of sublattice</td>
</tr>
<tr>
<td>GP2d_GetLowCoordX2</td>
<td>86</td>
<td>Get lower bound of sublattice</td>
</tr>
</tbody>
</table>
NAME   GP2d_ForAll - Iterator over whole sublattice.

C SYNOPSIS

#include <GP2d.h>
#define GP2d_ForAll(gp,x1,x2) \
   for(x1=GP2d_GetLowCoordX1(gp);x1<GP2d_GetHighCoordX1(gp);x1++) \
   for(x2=GP2d_GetLowCoordX2(gp);x2<GP2d_GetHighCoordX2(gp);x2++)

PARAMETERS

gp   The pointer to the struct containing informational parameters.

   x1 int index.

   x2 int index.

DESCRIPTION  This macro defines similar to the for statement in C an iterator over the 2-dimensional sublattice.

EXAMPLES   The program fills lattice with value '2'.

#include <stdio.h>
#include <GP2d.h>
void main(void)
{
    GP2d_InputParams params={{10,10},{2,2},{NULL,NULL},{1,1},sizeof(int)};
    GP2d *gp;
    int **lattice;
    int x1,x2;
    gp=GP2d_Initialize(params);
    lattice=(int **)GP2d_CreateLatt(gp);
    GP2d_ForAll(gp,x1,x2)
        lattice[x1][x2]=2;
    GP2d_UpdBndPbc(gp,lattice);
}
NAME GP2d_FStdReadLatt - Read lattice from a file. Simply read distributed lattice from file and distribute

C SYNOPSIS

#include <stdio.h>
#include <GP2d.h>
define GP2d_FStdReadLatt(gp,lattice,file,sourcetask) \ 
     GP2d_FReadLatt(lattice,file,GP2d_StdRead,0,sourcetask)

PARAMETERS

gp The pointer to the struct containing informational parameters.
lattice The pointer to the distributed lattice.
file The file descriptor the data is read.
sourcetask The task identifier of the task, that reads the file and distributes the data to all other tasks.

DESCRIPTION The function allows the restoring of a lattice configuration from one file. The boundary cells of the sublattices are not updated.

ERRORS

gp Not initialized by GP2d_Initialize.
lattice Not allocated by GP2d_CreateLatt.
file Invalid file pointer.
sourcetask Not in the range 0...GP2d_GetNumProcs(gp)[0]
     /* change toMPI!!! */ / * change to MPI!!! */

Program reads the contents of the lattice from a file.

EXAMPLES

#include <stdlib.h>
#include <stdio.h>
#include <GP2d.h>
define ROOTTASK 0
int main(void)
{
    GP2d_InputParams Params={100,100},{2,2},{NULL,NULL},{1,1},sizeof(float)};
    GP2d *gp;
    FILE *file;
    float **lattice;
gp = GP2d_Initialize(Params);
lattice = (float **) GP2d_CreateLatt(gp);
if (GP2d_GetMyTaskId(gp) == ROOTTASK) /* change to MPI!!! */
  if ((file = fopen("test.cfg","rb")) == NULL)
    {
      fprintf(stderr, "Error opening file.\n");
      exit(EXIT_FAILURE);
    }
GP2d_FStdReadLatt(gp, lattice, file, ROOTTASK);
if (GP2d_GetMyTaskId(gp) == ROOTTASK) /* change to MPI!!! */
  fclose(file);
GP2d_UpdBndPbc(gp, lattice);
exit(EXIT_SUCCESS);

RELATED INFORMATION

Functions: GP2d_FReadLatt, GP2d_FWriteLatt.
NAME  GP2d_FStdWriteLatt - Write lattice to a file. Simply collect and write distributed lattice to file

C SYNOPSIS

#include <stdio.h>
#include <GP2d.h>
#define GP2d_FStdWriteLatt(lattice,file,desttask) 
   GP2d_FWriteLatt(lattice,file,GP2d_StdWrite,0,desttask)

PARAMETERS

gp  The pointer to the struct containing informational parameters.
lattice  The pointer to the distributed lattice.
file  The file descriptor the data is written.
desttask  The task identifier of the task, that collects the data and writes the file.

DESCRIPTION  The function allows the storing of a lattice configuration to one file.

ERRORS

gp  Not initialized by GP2d_Initialize.
lattice  Not allocated by GP2d_CreateLatt.
file  Invalid file pointer.
desttask  Not in the range 0...GP2d_GetNumProcs(gp)[0]
   /* changetoMPI!!! */  / /  /  ×  GP2d_GetNumProcs(gp)[1] − 1. /* change to MPI!!! */

EXAMPLES  The program stores a float value lattice into a file.

#include <stdlib.h>
#include <stdio.h>
#include <GP2d.h>
#define ROOTTASK 0
int main(void)
{
   GP2d_InputParams Params={{100,100},{2,2},{NULL,NULL},{1,1},sizeof(float)};
   GP2d *gp;
   FILE *file;
   float **lattice;
   float cell=4.0;

   /* Additional code goes here... */
}
if((gp=GP2d_Initialize(Params))==NULL)
{
    fprintf(stderr,"Error initializing lattice.\n");
    exit(EXIT_FAILURE);
}

lattice=(float **)GP2d_CreateLatt(gp);
GP2d_InitLatt(gp,lattice,&cell);
if(GP2d_GetMyTaskId(gp)==ROOTTASK) /* change to MPI!!! */
    if((file=fopen("test.cfg","wb"))==NULL)
    {
        fprintf(stderr,"Error opening file.\n");
        exit(EXIT_FAILURE);
    }
GP2d_FStdWriteLatt(gp,lattice,file,ROOTTASK);
if(GP2d_GetMyTaskId(gp)==ROOTTASK) /* change to MPI!!! */
    fclose(file);
exit(EXIT_SUCCESS);

RELATED INFORMATION

Functions: GP2d_FReadLatt, GP2d_FWriteLatt.
NAME GP2d_GetBoundaryWidth - Get width of boundary.

C SYNOPSIS

#include <GP2d.h>
int* GP2d_GetBoundaryWidth(GP2d *gp);

PARAMETERS

gp The pointer to the struct containing informational parameters.

RETURN VALUE A pointer to an array containing the boundary width.

DESCRIPTION The functions returns a pointer to an array storing information about the boundary width for each space dimension.

NOTES This function is defined as a macro.

ERRORS

gp Not initialized by GP2d_Initialize.

EXAMPLES Program writes boundary widths on stdout.

#include <stdlib.h>
#include <stdio.h>
#include <GP2d.h>
#define ROOTTASK 0
int main(void)
{
    GP2d_InputParams params = {{100, 100}, {2, 2}, {NULL, NULL}, {1, 1}, sizeof(int)};
    GP2d *gp;
    if((gp = GP2d_Initialize(params)) == NULL)
    {
        fprintf(stderr, "Error initializing lattice\n");
        exit(EXIT_FAILURE);
    }
    if(GP2d_GetMyTaskId(gp) == ROOTTASK) /* change to MPI!!! */
    {
        printf("Boundary width in x1: %d\n", GP2d_GetBoundaryWidth(gp)[0],
               "Boundary width in x2: %d\n", GP2d_GetBoundaryWidth(gp)[1]);
        exit(EXIT_SUCCESS);
    }
}
NAME    GP2d_GetCellSize - Get cell size.

C SYNOPSIS

#include <GP2d.h>
int GP2d_GetCellSize(GP2d *gp);

PARAMETERS

gp    The pointer to the struct containing informational parameters.

RETURN VALUE    Size of one cell element.

DESCRIPTION    Function returns the size of one cell element in bytes.

ERRORS

gp Not initialized by GP2d_Initialize.

EXAMPLES    Program prints on stdout the size of lattice element.

#include <stdio.h>
#include <GP2d.h>
typedef struct
{
    int count;
    double coord[3];
}CELL_T;
int main(void)
{
    GP2d_InputParams params={{20,20},{2,2},{NULL,NULL},{1,1},sizeof(CELL_T)};
    GP2d *gp;
    gp=GP2d_Initialize(params);
    if(GP2d_GetMyTaskId(gp)==0) /* change to MPI!!! */
    {
        printf("Size(cell)=%d Bytes.\n",GP2d_GetCellSize(gp));
    }
}
NAME  GP2d_GetHighCoordX1 - Get upper boundary of sublattice.

C SYNOPSIS

#include <GP2d.h>
int GP2d_GetHighCoordX1(GP2d *gp);

PARAMETERS

gp  The pointer to the struct containing informational parameters.

RETURN VALUE  The upper bound in $x_1$-direction of the sublattice.

DESCRIPTION  The function returns the upper bound index of the local sublattice in $x_1$-direction.

NOTES  The function is implemented as a macro due to performance reasons.

ERRORS

gp  Not initialized by GP2d_Initialize.

EXAMPLES  The program fills the lattice with the sum of the cells coordinates.

#include <stdio.h>
#include <GP2d.h>
void main(void)
{
  GP2d_InputParams params={{10,10},{2,2},{NULL,NULL},{1,1},sizeof(int)};
  GP2d *gp;
  int **lattice;
  int x1,x2;
  gp=GP2d_Initialize(params);
  lattice=(int **)GP2d_CreateLatt(gp);
  for(x1=GP2d_GetLowCoordX1(gp);x1<GP2d_GetHighCoordX1(gp);x1++)
    for(x2=GP2d_GetLowCoordX2(gp);x2<GP2d_GetHighCoordX2(gp);x2++)
      lattice[x1][x2]=x1+x2;
  GP2d_UpdBndPbc(gp,lattice);
}

RELATED INFORMATION
NAME  GP2d_GetHighCoordX2 - Get upper bound of sublattice.

C SYNOPSIS

#include <GP2d.h>
int GP2d_GetHighCoordX2(GP2d *gp);

PARAMETERS

gp  The pointer to the struct containing informational parameters.

RETURN VALUE  The upper bound in $x_2$-direction of the sublattice.

DESCRIPTION  The function returns the upper bound index of the local sublattice in $x_2$-direction.

NOTES  The function is implemented as a macro due to performance reasons.

ERRORS

gp  Not initialized by GP2d_Initialize.

EXAMPLES

#include <stdio.h>
#include <GP2d.h>
void main(void)
{
    GP2d_InputParams params={{10,10},{2,2},{NULL,NULL},{1,1},sizeof(int)};
    GP2d *gp;
    int **lattice;
    int x,y;
    gp=GP2d_Initialize(params);
    gp=(int **)GP2d_CreateLatt(gp);
    for(x=GP2d_GetLowCoordX1(gp);x<GP2d_GetHighCoordX1(gp);x++)
        for(y=GP2d_GetLowCoordX2(gp);y<GP2d_GetLowCoordX2(gp);y++)
            lattice[x][y]=x+y;
    GP2d_UpdBndPbc(gp,lattice);
}

RELATED INFORMATION

NAME   GP2d_GetLattSize - Get size of lattice.

C SYNOPSIS

#include <GP2d.h>
int* GP2d_GetLattSize(GP2d *gp);

PARAMETERS

gp  The pointer to the struct containing informational parameters.

RETURN VALUE  A pointer to an array containing the lattice sizes.

DESCRIPTION  The functions returns a pointer to an array storing information about the lattice size for each space dimension.

NOTES  This function is defined as a macro.

ERRORS

gp  Not initialized by GP2d_Initialize.

EXAMPLES

#include <stdlib.h>
#include <stdio.h>
#include <GP2d.h>
#define ROOTTASK 0
int main(void)
{
    GP2d_InputParams params={{100,100},{2,2},{NULL,NULL},{1,1},sizeof(int)};
    GP2d *gp;
    if((gp=GP2d_Initialize(params))==NULL)
    {
        fprintf(stderr,"Error initializing lattice
        exit(EXIT_FAILURE);
    }
    if(GP2d_GetMyTaskId(gp)==ROOTTASK) /* change to MPI!!! */
    {
        printf("Lattice size in x1: %d\n"
            "Lattice size in x2: %d\n",
            GP2d_GetLattSize(gp)[0],
            GP2d_GetLattSize(gp)[1]);
        exit(EXIT_SUCCESS);
    }
NAME    GP2d_GetLowCoordX1 - Get lower boundary of sublattice.

C SYNOPSIS
#include <GP2d.h>
int GP2d_GetLowCoordX1(GP2d *gp);

PARAMETERS
    gp The pointer to the struct containing informational parameters.

RETURN VALUE    The lower bound in $x_1$-direction of the sublattice.

DESCRIPTION    The function returns the lower bound index of the local sublattice in $x_1$-direction.

NOTES    The function is implemented as a macro due to performance reasons.

ERRORS
    gp Not initialized by GP2d_Initialize.

EXAMPLES    The program fills the lattice with the sum of the cells coordinates.

#include <stdio.h>
#include <GP2d.h>
void main(void)
{
    GP2d_InputParams params={{10,10},{2,2},{NULL,NULL},{1,1},sizeof(int)};
    GP2d *gp;
    int **lattice;
    int x1,x2;
    gp=GP2d_Initialize(params);
    lattice=(int **)GP2d_CreateLatt(gp);
    for(x1=GP2d_GetLowCoordX1(gp);x1<GP2d_GetHighCoordX1(gp);x1++)
        for(x2=GP2d_GetLowCoordX2(gp);x2<GP2d_GetHighCoordX2(gp);x2++)
            lattice[x1][x2]=x1+x2;
    GP2d_UpdBndPbc(gp,lattice);
}

RELATED INFORMATION
NAME  GP2d_GetLowCoordX2 - Get lower boundary of sublattice.

C SYNSOPSIS

#include <GP2d.h>
int GP2d_GetLowCoordX2(GP2d *gp);

PARAMETERS

gp  The pointer to the struct containing informational parameters.

RETURN VALUE  The lower bound in \( x_2 \)-direction of the sublattice.

DESCRIPTION  The function returns the lower bound index of the local sublattice in \( x_2 \)-direction.

NOTES  The function is implemented as a macro due to performance reasons.

ERRORS

gp  Not initialized by GP2d_Initialize.

EXAMPLES  The program fills the lattice with the sum of the cells coordinates.

#include <stdio.h>
#include <GP2d.h>
void main(void)
{
    GP2d_InputParams params={{10,10},{2,2},{NULL,NULL},{1,1},sizeof(int)};
    GP2d *gp;
    int **lattice;
    int x1,x2;
    gp=GP2d_Initialize(params);
    lattice=(int **)GP2d_CreateLatt(gp);
    for(x1=GP2d_GetLowCoordX1(gp);x1<GP2d_GetHighCoordX1(gp);x1++)
        for(x2=GP2d_GetLowCoordX2(gp);x2<GP2d_GetHighCoordX2(gp);x2++)
            lattice[x1][x2]=x1+x2;
    GP2d_UpdBndPbc(gp,lattice);
}

RELATED INFORMATION

4 Appendix
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include "GP2d.h" /* Including prototypes and macros for GEOPAR */
#define uSecScale 1.0e-6 /* Microsecond conversions */
#define LATTFSIZE 400 /* Lattice size */
#define FILENAME "gol.cfg" /* Filename of configuration */
#define ROOTTASK 0 /* Task identifier, which performs I/O operations */

typedef enum {DEAD=0, ALIVE=1} STATE; /* Type definition for lattice elements */

double mrun(void)
{
    double ret;
    struct timeval tp;
    struct timezone tzp;
    gettimeofday(&tp,&tzp);
    ret = (((double)tp.tv_usec) * uSecScale) + (double)tp.tv_sec;
    return ret;
}

void initgol(GP2d *gp,STATE **ca)
{ /*sets initial state of the cellular automaton.*/
    int x1,x2;
    STATE cell;
    cell=DEAD;
    srand48(189128+GP2d_GetMyTaskId(gp));
    GP2d_ForAll(gp,x1,x2)
ca[x1][x2]=(drand48()<0.3) ? ALIVE : DEAD; /* random placement of cells */

GP2d_UpdBnd(gp, (void **)ca); /* Update the boundary cell with periodic */
   /* boundary conditions */
}
/* of 'initgol' */

void updategol(GP2d *gp, STATE **caold, STATE **canew)
/*****************************************************************************/
/**
**
**  u p d a t e  g o l
**
**  function performs an update of lattice according to the rules of the
**  "game of life" using an auxiliary lattice.
**
**
/*****************************************************************************/
{
    int x1, x2, count;

    GP2d_ForAll(gp, x1, x2) /* Iterator over whole sublattice */
    {
        count=caold[x1-1][x2-1]+ caold[x1][x2-1]+ caold[x1+1][x2-1]+
                  caold[x1-1][x2]+ caold[x1+1][x2]+ caold[x1-1][x2+1]+
                  caold[x1][x2+1]+ caold[x1+1][x2+1];

        /* Counting "living" neighbors of cell(x,y) */
        if(caold[x1][x2]==ALIVE)
            canew[x1][x2]=(count==2 || count==3) ? ALIVE : DEAD;
        else
            canew[x1][x2]=(count==3) ? ALIVE : DEAD;
    }

    GP2d_CopyLatt(gp,(void **)caold,(void **)canew);
    GP2d_UpdBnd(gp,(void **)caold); /* update periodic boundaries */
}
/* of 'updategol' */

void compresscolumn(GP2d *gp, char *buffer, const STATE *column)
{
    int x2;
    for(x2=GP2d_GetLowCoordX2(gp);x2<GP2d_GetHighCoordX2(gp);x2++)
        *buffer++=column[x2];
}
/* of 'compresscolumn' */

void readGol(GP2d *gp, STATE *column, char *buffer)
{  
    int x2;  
    for (x2 = GP2d_GetLowCoordX2(gp); x2 < GP2d_GetHighCoordX2(gp); x2++)  
        column[x2] = *buffer++;  
}  

void writegol(GP2d *gp, STATE **ca, FILE *file)  
{  
    GP2d_FWriteLatt(gp, (void **) ca, file, (GP2d_IOFunction) GP2d_DEFAULT,  
                    (GP2d_TransferFunction) compresscolumn, 1,  
                    ROOTTASK);  
}  /* of 'writegol' */  

int countgol(GP2d *gp, STATE **ca)  
/****************************************************************************/  
/****  
/**** countgol  
/**** Function counts the number of living cells of the distributed lattice.  
/**** The result of the global operation is returned in the ROOTTASK.  
/****  
/****  
/****************************************************************************/  
{  
    int x1, x2, count, global_count;  
    count = 0;  
    GP2d_ForAll(gp, x1, x2)  
        count += ca[x1][x2];  
    GP2d_ReduceData(gp, &count, &global_count, 1, MPI_INTEGER, ROOTTASK,  
                    MPI_SUM);  
    /* fprintf(stderr, "%d MPI_SUM, %d MPI_INT\n", MPI_SUM, MPI_INTEGER); */  
    #ifdef DEBUG  
    fprintf(stderr, "finished ReduceData at task \%d\n",  
            GP2d_GetMyTaskId(gp));  
    #endif  
    return global_count;  
}  /* of 'countgol' */  

void printcolumn(GP2d *gp, void *data, int x1, STATE *column)  
{  
    int x2;  
    printf("%3d ", x1);  
    for (x2 = 0; x2 < GP2d_GetLattSize(gp)[1]; x2++)  
        printf(column[x2] == ALIVE ? "*" : ".");  
    printf("\n");  
}  /* of 'printcolumn' */  

void println(GP2d *gp, STATE **ca, int iter)

90
if(GP2d_GetMyTaskId(gp)==ROOTTASK)
    printf("Iteration: %d\n",iter);

GP2d_Collect(gp,0,(void **)ca,(GP2d_CollectFunction)printcolumn,ROOTTASK);

if(GP2d_GetMyTaskId(gp)==ROOTTASK)
    printf("\n");

} /* of 'printgol' */

void printSubLatt(GP2d *gp, int **ca)
{
    int x, y;

    for(x=GP2d_GetLowCoordX1(gp)-GP2d_GetBoundaryWidth(gp)[0];
        x<GP2d_GetHighCoordX1(gp)+GP2d_GetBoundaryWidth(gp)[0];x++)
    {
        printf("%d%d | ", GP2d_GetMyTaskId(gp), x);
        for(y=GP2d_GetLowCoordX2(gp)-GP2d_GetBoundaryWidth(gp)[1];
            y<GP2d_GetHighCoordX2(gp)+GP2d_GetBoundaryWidth(gp)[1];y++)
        {
            printf(ca[x][y] == ALIVE ? "*" : ".");
        }
        printf("\n");
    }

} /* end of 'printSubLatt' */

int main(int argc,char **argv)
{
    GP2d *gp;

    struct
    {
        int n,step,size;
    } header;

    STATE **ca, **canew; /* Pointer to lattice */
    FILE *cfg; /* Output file for configurations */
    FILE *in;
    int i,count;
    double t1,t2;

    gp = (GP2d *) malloc(sizeof(GP2d));

    if (argc >= 7)
scanf(argv[5], "%d", &gp->InputParams.LattSize[0]);
scanf(argv[6], "%d", &gp->InputParams.LattSize[1]);

gp->InputParams.Reorder = TRUE;
gp->InputParams.Periods[0] = TRUE;
gp->InputParams.CellSize=sizeof(STATE); /* Size of lattice element */
gp->InputParams.Communicator=MPI_COMM_WORLD;

if (argc >= 5)
{
    scanf(argv[3], "%d", &gp->InputParams.NumProcs[0]);
    scanf(argv[4], "%d", &gp->InputParams.NumProcs[1]);
}
MPI_Init(&argc, &argv);

GP2d_Initialize(gp);
if(gp==NULL)
{
    fprintf(stderr,"Error allocating processors, program terminated.\n");
    return 1;
}

ca=(STATE **)GP2d_CreateLatt(gp, NULL);
ca=new=(STATE **)GP2d_CreateLatt(gp, NULL);

if(GP2d_GetMyTaskId(gp)==ROOTTASK)
{
    in=fopen("Cin2.dat","r");
    fread(&header, sizeof(header), 1, in);
    scanf(argv[1], "%d", &header.n);
    scanf(argv[2], "%d", &header.step);
    header.size = GP2d_GetLattSize(gp)[0];

    if(header.step)
    {
        /* Write header for configuration file */
        if((cfg=fopen(FILENAME,"wb"))==NULL)
        {
            fprintf(stderr,"Error creating file '%s'.\n",FILENAME);
            return 2;
        }
        fwrite(&header,sizeof(header),1, cfg);
    }
}
/* Distribute parameters to all tasks */
GP2d_BCastData(gp,&header,sizeof(header),ROOTTASK);

/* Set initial state of cellular automaton */
/* initgol(gp, ca); */
GP2d_FReadLatt(gp, ca, in, (GP2d_IOFunction) GP2d_DEFAULT,
               (GP2d_TransferFunction) readGol, 1, 0);
GP2d_UpdBnd(gp, ca);
t1=mrun();
if (GP2d_GetMyTaskId(gp) == ROOTASK )
   fclose(in);

for(i=1;i<=header.n;i++)
{
   updategol(gp, ca, canew); /* Update cellular automaton */
   if(header.step && ((i % header.step) == 0))
   {
      printfgol(gp,ca,i);
      count=countgol(gp,ca); /* Count number of "living" */
      /* cells */
      writegol(gp,ca,cfg); /* Write configuration of CA into file */
      if(GP2d_GetMyTaskId(gp)==ROOTASK)
         fprintf(stderr,"i=%d,count=%d\n",i,count);
   }
}
if(GP2d_GetMyTaskId(gp)==ROOTASK)
{
   t2=mrun();
   printf("Time= %.3f sec.\n",t2-t1);
}
if(header.step && GP2d_GetMyTaskId(gp)==ROOTASK)
   fclose(cfg);
return 0;
} /* of 'main' */