## Contents

1 BSPonMPI 1
   1.1 Introduction ................................. 1
   1.2 Architecture ............................... 1
   1.3 License ...................................... 10

2 BSPonMPI Hierarchical Index 11
   2.1 BSPonMPI Class Hierarchy .................... 11

3 BSPonMPI Data Structure Index 13
   3.1 BSPonMPI Data Structures .................... 13

4 BSPonMPI File Index 15
   4.1 BSPonMPI File List ........................... 15

5 BSPonMPI Page Index 17
   5.1 BSPonMPI Related Pages ...................... 17

6 BSPonMPI Data Structure Documentation 19
   6.1 _BSPObject Struct Reference .................. 19
   6.2 _ExpandableTable Struct Reference .......... 23
   6.3 _VarSzInfo Union Reference .................. 25
   6.4 DelivInfo Struct Reference ................... 27
   6.5 MemRegInfo Struct Reference ................ 28
   6.6 MesgQInfo Struct Reference ................... 30
   6.7 PopRegObject Struct Reference ............... 32
   6.8 PushRegObject Struct Reference .............. 33
   6.9 PutObject Struct Reference .................. 34
Chapter 1

BSPonMPI

BSPonMPI is an implementation of the BSPlib standard on top of MPI 1.1

1.1 Introduction

You should read this document if you want to make changes to BSPonMPI or if you want to write your own BSPlib implementation and you need some inspiration. If you just want to use BSPonMPI, you should read the README file (included with the BSPonMPI package) first. In all cases I assume that the reader is familiar with the BSPlib standard.

This document contains

- A brief overview of the architecture of BSPonMPI
- Short descriptions of every function and data structure
- Detailed explanation of some non-standard (weird) solutions
- A lot of pictures and cross references

If you want to change something in BSPonMPI, you should read the Architecture description and the detailed descriptions of the code which will be affected by your changes. If you just need inspiration, you only need to read the Architecture description.

1.2 Architecture

1.2.1 Requirements

The mission is to make a BSPlib implementation which will always be faster than the Oxford BSP Toolset on top of MPI and runs on all MPI enabled computer platforms.
Until now this performance mission has not been achieved. BSPonMPI is still a bit slower when executing `bsp_put()` but it is faster on executing `bsp_get()` and `bsp_send()`.

### 1.2.2 General Idea

The BSPlib standard is designed to make programming using the Bulk Synchronous Parallel (BSP) model easier. The BSP model prescribes that you cut up your program in computation and communication supersteps. BSPlib helps you by providing a function `bsp_sync()` to execute an entire communication superstep. Communication is gathered during a computation superstep using three functions: `bsp_put()`, `bsp_get()` and `bsp_send()`. To put it very simple: A BSPlib implementation is a

- buffer
- having three functions to fill it: `bsp_put()`, `bsp_get()` and `bsp_send()`, and
- one function to empty it: `bsp_sync()`

Actually it is not that simple: A `bsp_get()` needs some action to be taken on the remote processor and expects some data to be returned. This very simple model can be repaired by seeing that there are not only data deliveries (`bsp_put()` and `bsp_send()`) but also data requests (`bsp_get()`). Therefore we may conclude that any BSPlib implementation consists of two buffers: one request buffer and one delivery buffer. Note that I ignore the two unbuffered communication routines `bsp_hpput()` and `bsp_hpget()`. These are currently implemented as their buffered counterparts.

Let me now restate what a BSPlib implementation is:

- Two communication buffers: One data request and one data delivery buffer.
- Three functions to fill them: `bsp_get()` which adds a request to the data request buffer, and `bsp_send()` and `bsp_put()` which add a delivery to the data delivery buffer.
- One function to empty them: `bsp_sync()`

### 1.2.3 Design Choices

The idea is to use the bulk communication procedures of MPI. The two most BSP like MPI functions are: `MPI_Alltoall()` and `MPI_Alltoallv()`. Nowadays computer manufacturers implement efficient MPI 1.1 libraries. Any MPI enabled machine has these functions and it is very probable that they are optimal. You may ask yourself why I don’t use the DRMA procedures of MPI-2. They may be very fast on DRMA enabled machines. However MPI-2 is not yet widely available and will therefore limit the use of this library.

In order to effectively use these communication procedures we need that the surrounding code is very fast and portable. The choices I made are

1. Use ANSI C 99 as programming language in combination with the GNU auto-tools.
1.2 Architecture

2. Use an object oriented programming style.

3. Collect all communication and transmit them using the least possible calls to MPI_Alltoallv.

4. Use arrays to implement communication buffers. They can be used in a number of contexts, e.g. as a queue and as parameter of the MPI_Alltoallv() function.

5. Allocate memory for these buffers only once (at the start of the program) and double the allocated memory if the buffer proves to be too small. Using this strategy the number of calls to malloc() remains very small.

6. Minimise the use of branch statements (if, switch). I discovered this too late and I expect that more performance can be gained by rethinking the main data structure

7. Try to make the optimising as easy as possible for a C compiler, but only use ANSI C constructions. Examples: restrict, inline, static, memory alignment, rather dereference a pointer than use memcpy(), etc...

Note:
When writing this documentation I discovered a lot of illogical constructions. This is caused by using trial and error design of this library. Each time when I discovered a way to improve performance I had to weigh the implementation time cost against performance improvement. Therefore sometimes the overall architecture does not comply to some design choices, some of which I made during experimenting and tweaking of the library.

1.2.4 Implementation

1.2.4.1 Object Oriented Programming in ANSI C

Though I used ANSI C as programming language, I tried to program in an object oriented fashion. To translate classes and inheritance into C constructs, I used struct's, union's and enum's, e.g.: the C++ code

```c
class A { int x, y;};
class B: A { int a, b;};
class C: A { int c, d;};

void foo()
{
  A a; a.x = 0;
  B b; b.a = 1; b.x = 0;
  C c; c.d = 2; c.y = 1;
}
```

translates into something like

```c
typedef enum { B_type, C_type } class_type;
typedef struct { int a, b; } B_info;
typedef struct { int c, d; } C_info;
typedef union { B_info b; C_info c;} class_info;
```

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
typedef struct { int x, y; class_info info; class_type type;} A;

void foo()
{
    A a; a.x = 0;
    A b; b.type = B_type; b.b.a=1; b.x = 0;
    A c; c.type = C_type; c.c.d=2; c.y = 1;
}

Member functions are translated into className_functionName( classref *, ...). For example

class A
{
    int x;
    int foo() const
    {
        return x;
    }
};

translates into

typedef struct {int x;} A;
int a_foo( const A * restrict a)
{
    return a->x;
}

1.2.5 Communication Buffer

Because BSPlib is essentially a communication buffer, the performance of the library heavily depends on its data structure. As primitive communication buffer I designed ExpandableTable. This is essentially a one-dimensional array which is subdivided in equally sized blocks. I take as many blocks as there are processors and assign each block to a processor. When you look at these blocks as columns, you will get a table where each column corresponds to a processor. Putting data in a column, marks that data as 'received from' or 'to be sent to' the corresponding processor. This way, the array can be passed as a parameter of the MPI_Alltoallv() function.
Each column is again subdivided in slots. Accessing arbitrary bytes may be very expensive on some architectures or impossible. Therefore I try to use aligned addresses by subdividing each column in slots. Usually I take `slot_size` equal to the size of a `struct` which I use as basic array element. The size of the entire array is equal to `nprocs x rows x slot_size`, where `rows` is the number of slots in a column.

When data is added to a column the variable `used_slot_count` is incremented with the number of slots occupied by the new data. Additionally the variable `count` is incremented with 1. An example is shown below:

![Diagram](image-url)
When one tries to add data to a column but there is not enough space available, new space is allocated. The new space is three times the size of the already allocated space. Again this array is subdivided in equally sized blocks. The old data is copied to the new locations and the new data is added. Note that the size of an ExpandableTable can only increase. This way only a very limited number of calls to malloc() are necessary. ExpandableTable and its member functions are declared in bsp_exptable.c and bsp_exptable.h.

This data structure serves as a building block for the two communication buffers: RequestTable and DeliveryTable. They both have slightly different needs. RequestTable only has to handle data requests from other processors. Each data request is of a fixed size. Therefore implementing RequestTable will be rather straightforward. For details I refer to bsp_reqtable.h and bsp_reqtable.c. On the other hand DeliveryTable has to handle data deliveries which may differ in size. The implementation of DeliveryTable is a bit different; I use this buffer not only for communication of bsp_put(), bsp_send() and the data delivery part of a bsp_get(), but also for actions which have to be carried out during a bsp_sync(), e.g.: bsp_set_tagsize(), etc... for details see bsp_delivtable.h and bsp_delivtable.c.

BSPlib also provides a way to address remote memory locations and a queue of received messages. These two can also be modelled in a fixed size element array and a variable size element array, respectively. Therefore two abstract classes are introduced: FixedElSizeTable and VarElSizeTable. We get the following UML class diagram.
1.2 Architecture

A small UML legenda is shown below.

Small UML Legenda

Figure 1.3: Legenda of UML class diagram and UML Sequence Diagram

1.2.6 A Sequence Diagram

Until now the explanation may still be a bit abstract. Below the source code and its sequence diagram of a simple BSP program are shown. It shows how processors and objects collaborate over time.

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

void spmd_part()
```
{ int a=1, b=2, c=3;
 bsp_begin(2);
 bsp_push_reg(&a, sizeof(int));
 bsp_sync();

 if (bsp_pid() == 0)
 { bsp_put(1, &b, &a, 0, sizeof(int));
   bsp_get(1, &a, 0, &c, sizeof(int));
   bsp_send(1, NULL, "some text", 10);
 }
 bsp_sync();
 bsp_pop_reg(&a);
 bsp_sync();
 bsp_end();

 int main(int argc, char *argv)
 { bsp_init(&spmd_part, argc, argv);
   printf("First perform some sequential code\n");
   spmd_part();
   printf("Finally perform some sequential code\n");
   return 0;
 }
Figure 1.4: Sequence diagram of the example above
1.2.7 Room For Improvement

There are still things which are not being taken care of in an optimal way. Ideas to improve performance are:

1. Merge the two MPI_Alltoall()’s of RequestTable and DeliveryTable which exchange buffer sizes. Or make it an exception to exchange buffer sizes: use MPI_Allgather() to determine whether buffer sizes need to be communicated. This function may be less expensive.

2. subdivide every column in ExpandableTable, where each part only holds elements of one kind, e.g.: every column has a send, put, other (=settag, pushreg, popreg, etc...) block. This avoids branches in tight loops (in deliveryTable_execute() )

3. Align every data element

4. Implement the MessageQueue s.t. copying memory from DeliveryTable becomes superfluous.

5. When expanding a buffer, do it only for a specific processor / buffer. This increases performance on non full h-relation

6. Implement the memoryRegister_find such that it is O(1) in stead of O(n) (e.g. hash on least significant bits)

1.3 License

BSPonMPI. This is an implementation of the BSPlib standard on top of MPI. Copyright ©2006 Wijnand J. Suijlen

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
Chapter 2

BSPonMPI Hierarchical Index

2.1 BSPonMPI Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

_BSPObject ....................................................... 19
ExpandableTable ................................................. 23
_VarSzInfo ....................................................... 25
DelivInfo ......................................................... 27
MemRegInfo ...................................................... 28
MsgQInfo ......................................................... 30
PopRegObject .................................................... 32
PushRegObject .................................................. 33
PutObject ......................................................... 34
ReqElement ...................................................... 35
SendObject ....................................................... 37
SetTagObject .................................................... 39
SpecInfo ......................................................... 40
VarSizeElement ................................................... 41
Chapter 3

BSPOnMPI Data Structure Index

3.1 BSPOnMPI Data Structures

Here are the data structures with brief descriptions:

- _BSPObject (Global variables used in bsp.c) ........................................... 19
- _ExpandableTable (Table with nprocs number columns and rows number of rows, where each row has a height of slot_size bytes) .............. 23
- _VarSzInfo (Data structure for additional info in a VarSizeElement) .......... 25
- DelivInfo (Additional data for a DeliveryTable) ..................................... 27
- MemRegInfo (Additional data needed by a MemoryRegister object) .......... 28
- MsgQInfo (Additional data needed by a MessageQueue object) ............... 30
- PopRegObject (Additional info for a bsp_pop_Reg() ) ............................ 32
- PushRegObject (Additional info for a bsp_push_reg() ) ......................... 33
- PutObject (Additional info for a bsp_put() ) ........................................ 34
- ReqElement (Data element stored in a RequestTable object) ................... 35
- SendObject (Additional info for a bsp_send() ) ..................................... 37
- SetTagObject (Additional info for a bsp_set_tagsize() ) ......................... 39
- SpecInfo (Datastructure storing specific info) ....................................... 40
- VarSizeElement (Data element stored in a VarElSizeTable (e.g)) ............. 41
Chapter 4

BSPonMPI File Index

4.1 BSPonMPI File List

Here is a list of all files with brief descriptions:

- bsp.c (Implements the BSPlib primitives) ............................................ 43
- bsp_abort.c (Implements the internal abort function: bsp_intern_abort()) .... 58
- bsp_abort.h (Defines the prototype of an ‘abort’ function for internal usage:
  bsp_intern_abort()) ........................................................................... 60
- bsp_alloc.h (Wrapper functions for malloc(), calloc() and free()) ........... 63
- bsp_delivtable.c (Implements the methods on a DeliveryTable object) ....... 67
- bsp_delivtable.h (Defines the prototypes of the methods on a DeliveryTable
  object) ............................................................................................... 69
- bsp_extable.c (Implements the communication method on an Expandable-
  Table) ................................................................................................. 75
- bsp_extable.h (Defines struct’s, prototypes and inlined functions of the
  ExpandableTable, FixedElSizeTable and VarElSizeTable ‘classes’
  and some struct’s of DeliveryTable, RequestTable, Memory-
  Register and MessageQueue) ................................................................ 77
- bsp_memreg.c (Implements methods on MemoryRegister) ......................... 86
- bsp_memreg.h (Defines prototypes of methods on MemoryRegister) .......... 91
- bsp_mesgqueue.h (Defines methods on MessageQueue) .............................. 94
- bsp_private.h (Global variable which should not be available to the outside
  world) .................................................................................................... 101
- bsp_reqtable.c (Implement the ‘execute’ method on a RequestTable) ........ 103
- bsp_reqtable.h (Defines prototypes and implementations of methods on
  RequestTable) ...................................................................................... 105
Chapter 5

BSPonMPI Page Index

5.1  BSPonMPI Related Pages

Here is a list of all related documentation pages:

  Bug List .......................................................... 109
Chapter 6

BSPonMPI Data Structure Documentation

6.1 _BSPObject Struct Reference

global variables used in bsp.c
#include <bsp_private.h>

Data Fields

• double begintime
  start time in bsp_begin()

• int nprocs
  number of processors

• int rank
  rank of this process

• MPI_Comm communicator
  MPI Communicator of the BSP program.

• ExpandableTable delivery_table
  Message combining table for delivery of data, i.e.

• ExpandableTable delivery_received_table
  Table in which received data is stored to be executed later.

• ExpandableTable request_table
  Message combining table for just bsp_get() requests.
• ExpandableTable request_received_table
  Table in which received requests are stored to be executed later.

• ExpandableTable memory_register
  Memory register.

• ExpandableTable message_queue
  Message queue.

6.1.1 Detailed Description

global variables used in bsp.c
Definition at line 41 of file bsp_private.h.

6.1.2 Field Documentation

6.1.2.1 double _BSPObject::begintime

start time in bsp_begin()
Definition at line 43 of file bsp_private.h.
Referenced by bsp_begin(), and bsp_time().

6.1.2.2 MPI_Comm _BSPObject::communicator

MPI Communicator of the BSP program.
Definition at line 47 of file bsp_private.h.
Referenced by bsp_begin(), and bsp_sync().

6.1.2.3 ExpandableTable _BSPObject::delivery_received_table

Table in which received data is stored to be executed later.
Definition at line 53 of file bsp_private.h.
Referenced by bsp_begin(), bsp_end(), and bsp_sync().

6.1.2.4 ExpandableTable _BSPObject::delivery_table

Message combining table for delivery of data, i.e.
  : bsp_put(), bsp_send() and the delivery of bsp_get()
Definition at line 51 of file bsp_private.h.
6.1 _BSPObject Struct Reference

Referenced by bsp_begin(), bsp_end(), bsp_pop_reg(), bsp_push_reg(), bsp_put(), bsp_send(), bsp_set_tagsize(), and bsp_sync().

6.1.2.5 **ExpandableTable _BSPObject::memory_register**

Memory register.
Tracks registered variables and memory locations to be used in DRMA operations, i.e.: bsp_get() and bsp_put()

Definition at line 60 of file bsp_private.h.
Referenced by bsp_begin(), bsp_end(), bsp_get(), bsp_put(), and bsp_sync().

6.1.2.6 **ExpandableTable _BSPObject::message_queue**

Message queue.
Contains all received data initiated by a bsp_send() on some processor.

Definition at line 63 of file bsp_private.h.
Referenced by bsp_begin(), bsp_end(), bsp_get_tag(), bsp_hpmove(), bsp_move(), bsp_qsize(), bsp_send(), bsp_set_tagsize(), and bsp_sync().

6.1.2.7 **int _BSPObject::nprocs**

count of processors

Definition at line 44 of file bsp_private.h.
Referenced by bsp_begin(), bsp_init(), bsp_nprocs(), and bsp_push_reg().

6.1.2.8 **int _BSPObject::rank**

rank of this process

Definition at line 45 of file bsp_private.h.
Referenced by bsp_begin(), bsp_init(), bsp_pid(), bsp_pop_reg(), and bsp_set_tagsize().

6.1.2.9 **ExpandableTable _BSPObject::request_received_table**

Table in which received requests are stored to be executed later.
Definition at line 57 of file bsp_private.h.
Referenced by bsp_begin(), bsp_end(), and bsp_sync().

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
6.1.2.10  ExpandableTable_BSPObject::request_table

Message combining table for just \texttt{bsp\_get()} requests.
Definition at line 55 of file \texttt{bsp\_private.h}.
Referenced by \texttt{bsp\_begin()}, \texttt{bsp\_end()}, \texttt{bsp\_get()}, and \texttt{bsp\_sync()}.
The documentation for this struct was generated from the following file:

- \texttt{bsp\_private.h}
6.2 _ExpandableTable Struct Reference

A table with `nprocs` number columns and `rows` number of rows, where each row has a height of `slot_size` bytes.

```c
#include <bsp_exptable.h>
```

## Data Fields

- `int nprocs`
  
  Number of processors, or columns

- `int rows`
  
  Number of elements (VarSizeElement, MemRegElement, ReqElement) allocated per processor

- `int *restrict count`
  
  Number of elements in table per column

- `int *restrict used_slot_count`
  
  Number of slots used in table per column

- `int slot_size`
  
  Size of the slots

- `SpecInfo info`
  
  Specific information dependent which class is derived from this on.

- `char * data`
  
  Pointer to actual table

### 6.2.1 Detailed Description

A table with `nprocs` number columns and `rows` number of rows, where each row has a height of `slot_size` bytes.

This table can be communicated to the other processors by means of `expandableTable_->comm()`: each columns is send to processors with rank equal to the column number.

Definition at line 204 of file `bsp_exptable.h`.

### 6.2.2 Field Documentation

#### 6.2.2.1 `int * restrict _ExpandableTable::count`

Number of elements in table per column.

---

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
Definition at line 210 of file bsp_exptable.h.

6.2.2.2 char _ExpandableTable::data

pointer to actual table
Definition at line 220 of file bsp_exptable.h.

6.2.2.3 union SpecInfo _ExpandableTable::info

Specific information dependent which class is derived from this on.
Definition at line 217 of file bsp_exptable.h.

6.2.2.4 int _ExpandableTable::nprocs

number of processors, or columns
Definition at line 206 of file bsp_exptable.h.

6.2.2.5 int _ExpandableTable::rows

number of elements (VarSizeElement, MemRegElement, ReqElement) allocated per processor
Definition at line 207 of file bsp_exptable.h.

6.2.2.6 int _ExpandableTable::slot_size

size of the slots
Definition at line 214 of file bsp_exptable.h.

6.2.2.7 int restrict _ExpandableTable::used_slot_count

number of slots used in table per column
Definition at line 212 of file bsp_exptable.h.

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

• bsp_exptable.h
6.3 _VarSzInfo Union Reference

Data structure for additional info in a VarSizeElement.

```
#include <bsp_exptable.h>
```

Data Fields

- PutObject put
- SendObject send
- PushRegObject push
- PopRegObject pop
- SetTagObject settag

6.3.1 Detailed Description

Data structure for additional info in a VarSizeElement.
Definition at line 157 of file bsp_exptable.h.

6.3.2 Field Documentation

6.3.2.1 PopRegObject _VarSzInfo::pop

Definition at line 162 of file bsp_exptable.h.
Referenced by bsp_pop_reg().

6.3.2.2 PushRegObject _VarSzInfo::push

Definition at line 161 of file bsp_exptable.h.
Referenced by bsp_push_reg().

6.3.2.3 PutObject _VarSzInfo::put

Definition at line 159 of file bsp_exptable.h.
Referenced by deliveryTable_pushPut(), and exec_put().

6.3.2.4 SendObject _VarSzInfo::send

Definition at line 160 of file bsp_exptable.h.
Referenced by deliveryTable_pushSend().
6.3.2.5  SetTagObject _VarSzInfo::settag

Definition at line 163 of file bsp_exptable.h.
Referenced by bsp_set_tagsize().
The documentation for this union was generated from the following file:

- bsp_exptable.h
6.4 DelivInfo Struct Reference

Additional data for a DeliveryTable.

```c
#include <bsp_exptable.h>
```

**Data Fields**

- `VarSizeElement **restrict latest_pushed_element`
  
  *array of pointers which point to the last element pushed.*

6.4.1 Detailed Description

Additional data for a DeliveryTable.

Definition at line 181 of file `bsp_exptable.h`.

6.4.2 Field Documentation

6.4.2.1 `VarSizeElement ** restrict DelivInfo::latest_pushed_element`

array of pointers which point to the last element pushed.

Is necessary for combining `bsp_put()` and `bsp_send()`

Definition at line 185 of file `bsp_exptable.h`.

Referenced by `deliveryTable_expand()`, and `deliveryTable_initialize()`.

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

- `bsp_exptable.h`
6.5 MemRegInfo Struct Reference

Additional data needed by a MemoryRegister object.

```c
#include <bsp_exptable.h>
```

### Data Fields

- `int numremov`
  
  number of pointers popped

- `int *restrict removed`
  
  boolean array of popped pointers

- `int memoized_src_proc`
  
  Rank of this processor.

- `const MemRegElement * memoized_data_iter`
- `const MemRegElement * memoized_end`
- `int memoized_srccol`

#### 6.5.1 Detailed Description

Additional data needed by a MemoryRegister object.

Definition at line 91 of file bsp_exptable.h.

#### 6.5.2 Field Documentation

##### 6.5.2.1 `const MemRegElement* MemRegInfo::memoized_data_iter`

Definition at line 96 of file bsp_exptable.h.

Referenced by memoryRegister_expand(), and memoryRegister_initialize().

##### 6.5.2.2 `const MemRegElement* MemRegInfo::memoized_end`

Definition at line 97 of file bsp_exptable.h.

Referenced by memoryRegister_expand(), and memoryRegister_initialize().

##### 6.5.2.3 `int MemRegInfo::memoized_src_proc`

Rank of this processor.

Definition at line 95 of file bsp_exptable.h.

Referenced by memoryRegister_expand(), and memoryRegister_initialize().
6.5 MemRegInfo Struct Reference

6.5.2.4 int MemRegInfo::memoized_srccol

Definition at line 98 of file bsp_exptable.h.
Referenced by memoryRegister_expand(), and memoryRegister_initialize().

6.5.2.5 int MemRegInfo::numremov

number of pointers popped
Definition at line 93 of file bsp_exptable.h.
Referenced by memoryRegister_expand(), and memoryRegister_initialize().

6.5.2.6 int restrict MemRegInfo::removed

boolean array of popped pointers
Definition at line 94 of file bsp_exptable.h.
Referenced by memoryRegister_expand(), and memoryRegister_initialize().
The documentation for this struct was generated from the following file:

- bsp_exptable.h
6.6  MesgQInfo Struct Reference

Additional data needed by a MessageQueue object.
#include <bsp_exptable.h>

Data Fields

- int  tag_size
  current tag size

- int  slot_offset
  points to VarSizeElement describing current sendobject

- int  inslot_offset
  number of next tag,payload pair of the current sendobject

- int  n_mesg
  number of messages in queue

- int  n_mesg_deq
  number of dequeued messages

- int  accum_size
  summed payload_sizes of all elements

6.6.1  Detailed Description

Additional data needed by a MessageQueue object.
Definition at line 73 of file bsp_exptable.h.

6.6.2  Field Documentation

6.6.2.1  int  MesgQInfo::accum_size

summed payload_sizes of all elements
Definition at line 80 of file bsp_exptable.h.
Referenced by messageQueue_initialize().

6.6.2.2  int  MesgQInfo::inslot_offset

number of next tag,payload pair of the current sendobject
6.6 MesgQInfo Struct Reference

Definition at line 77 of file bsp_exptable.h. Referenced by messageQueue_initialize().

6.6.2.3 int MesgQInfo::n_mesg

number of messages in queue
Definition at line 78 of file bsp_exptable.h. Referenced by messageQueue_initialize().

6.6.2.4 int MesgQInfo::n_mesg_deq

number of dequeued messages
Definition at line 79 of file bsp_exptable.h. Referenced by messageQueue_initialize().

6.6.2.5 int MesgQInfo::slot_offset

points to VarSizeElement describing current sendobject
Definition at line 76 of file bsp_exptable.h. Referenced by messageQueue_initialize().

6.6.2.6 int MesgQInfo::tag_size

current tag size
Definition at line 75 of file bsp_exptable.h. Referenced by messageQueueInitialize().

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

- bsp_exptable.h
6.7 PopRegObject Struct Reference

Additional info for a bsp_pop_Reg().
#include <bsp_exptable.h>

Data Fields

• const char * address

6.7.1 Detailed Description

Additional info for a bsp_pop_Reg().
Definition at line 145 of file bsp_exptable.h.

6.7.2 Field Documentation

6.7.2.1 const char* PopRegObject::address

Definition at line 147 of file bsp_exptable.h.
Referenced by bsp_pop_reg().
The documentation for this struct was generated from the following file:

• bsp_exptable.h
6.8 PushRegObject Struct Reference

Additional info for a bsp_push_reg().

#include <bsp_exptable.h>

Data Fields

• const char * address

6.8.1 Detailed Description

Additional info for a bsp_push_reg().
Definition at line 139 of file bsp_exptable.h.

6.8.2 Field Documentation

6.8.2.1 const char* PushRegObject::address

Definition at line 141 of file bsp_exptable.h.
Referenced by bsp_push_reg().
The documentation for this struct was generated from the following file:

• bsp_exptable.h
6.9 PutObject Struct Reference

additional info for a bsp_put()
#include <bsp_exptable.h>

Data Fields

- int item_count
  Number of puts packed in this big ’bsp_put’.

- int item_size
  Size of each bsp_put.

6.9.1 Detailed Description

additional info for a bsp_put()
Definition at line 124 of file bsp_exptable.h.

6.9.2 Field Documentation

6.9.2.1 int PutObject::item_count

Number of puts packed in this big ’bsp_put’.
Definition at line 126 of file bsp_exptable.h.
Referenced by deliveryTable_pushPut(), and exec_put().

6.9.2.2 int PutObject::item_size

Size of each bsp_put.
Definition at line 127 of file bsp_exptable.h.
Referenced by deliveryTable_pushPut(), and exec_put().

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

- bsp_exptable.h
6.10 ReqElement Struct Reference

Data element stored in a RequestTable object.
#include <bsp_exptable.h>

Data Fields

- int size
- int offset
  
  size and offset of requested data

- char * src
- char * dst
  
  memory locations in src and dst procs

6.10.1 Detailed Description

Data element stored in a RequestTable object.
A RequestTable stores only ‘bsp_get()’ operations
Definition at line 111 of file bsp_exptable.h.

6.10.2 Field Documentation

6.10.2.1 char * ReqElement::dst

memory locations in src and dst procs
Definition at line 114 of file bsp_exptable.h.
Referenced by bsp_get().

6.10.2.2 int ReqElement::offset

size and offset of requested data
Definition at line 113 of file bsp_exptable.h.
Referenced by bsp_get().

6.10.2.3 int ReqElement::size

Definition at line 113 of file bsp_exptable.h.
Referenced by bsp_get().

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
6.10.2.4 char * ReqElement::src

Definition at line 114 of file bsp_exptable.h.

Referenced by bsp_get().

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

- bsp_exptable.h
6.11 SendObject Struct Reference

additional info for a bsp_send()
#include <bsp_exptable.h>

Data Fields

- int tag_size
  Size of the tags.

- int payload_size
  Size of the payloads.

- int item_count
  Number of sends packined in this big 'bsp_send()'.

6.11.1 Detailed Description

additional info for a bsp_send()
Definition at line 131 of file bsp_exptable.h.

6.11.2 Field Documentation

6.11.2.1 int SendObject::item_count

Number of sends packined in this big 'bsp_send()'.
Definition at line 135 of file bsp_exptable.h.
Referenced by deliveryTable_pushSend().

6.11.2.2 int SendObject::payload_size

Size of the payloads.
Definition at line 134 of file bsp_exptable.h.
Referenced by deliveryTable_pushSend().

6.11.2.3 int SendObject::tag_size

Size of the tags.
Definition at line 133 of file bsp_exptable.h.
Referenced by deliveryTable_pushSend().
The documentation for this struct was generated from the following file:

- bsp_exptable.h
6.12 SetTagObject Struct Reference

Additional info for a `bsp_set_tagsize()`.
```
#include <bsp_exptable.h>
```

Data Fields

- `int tag_size`

6.12.1 Detailed Description

Additional info for a `bsp_set_tagsize()`.
Definition at line 151 of file `bsp_exptable.h`.

6.12.2 Field Documentation

6.12.2.1 `int SetTagObject::tag_size`

Definition at line 153 of file `bsp_exptable.h`.
Referenced by `bsp_set_tagsize()`.
The documentation for this struct was generated from the following file:

- `bsp_exptable.h`
6.13 SpecInfo Union Reference

Datastructure storing specific info.

```c
#include <bsp_exptable.h>
```

### Data Fields

- MemRegInfo reg
- DelivInfo deliv
- MesgQInfo mesgq

#### 6.13.1 Detailed Description

Datastructure storing specific info.

Definition at line 189 of file bsp_exptable.h.

#### 6.13.2 Field Documentation

6.13.2.1 DelivInfo SpecInfo::deliv

Definition at line 192 of file bsp_exptable.h.

Referenced by deliveryTable_expand(), and deliveryTable_initialize().

6.13.2.2 MesgQInfo SpecInfo::mesgq

Definition at line 194 of file bsp_exptable.h.

Referenced by messageQueue_initialize().

6.13.2.3 MemRegInfo SpecInfo::reg

Definition at line 191 of file bsp_exptable.h.

Referenced by memoryRegister_expand(), and memoryRegister_initialize().

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

- bsp_exptable.h
**6.14 VarSizeElement Struct Reference**

Data element stored in a VarElSizeTable (e.g.

```c
#include <bsp_exptable.h>
```

**Data Fields**

- **int size**
  
  *Size of the variable-length data stored after this structure.*

- **ItemType type**
  
  *info and data type*

- **_VarSzInfo info**
  
  *Specific info of an element.*

- **const char *restrict data**
  
  *pointer to the data. when stored in a table this pointer loses its meaning and is set to NULL.*

**6.14.1 Detailed Description**

Data element stored in a VarElSizeTable (e.g. a DeliveryTable or a MessageQueue)

Definition at line 168 of file bsp_exptable.h.

**6.14.2 Field Documentation**

**6.14.2.1 const char* restrict VarSizeElement::data**

pointer to the data, when stored in a table this pointer loses its meaning and is set to NULL.

Definition at line 176 of file bsp_exptable.h.

Referenced by bsp_pop_reg(), bsp_push_reg(), bsp_set_tagsize(), deliveryTable_.expand(), deliveryTable_pushPut(), deliveryTable_pushSend(), and varElSizeTable_.push().

**6.14.2.2 union _VarSzInfo VarSizeElement::info**

Specific info of an element.

Definition at line 172 of file bsp_exptable.h.
Referenced by bsp_pop_reg(), bsp_push_reg(), bsp_set_tagsize(), deliveryTable_pushPut(), deliveryTable_pushSend(), and exec_put().

6.14.2.3 int VarSizeElement::size

Size of the variable-length data stored after this structure.
Definition at line 170 of file bsp_exptable.h.
Referenced by bsp_pop_reg(), bsp_push_reg(), bsp_set_tagsize(), deliveryTable_pushPut(), deliveryTable_pushSend(), and varElSizeTable_push().

6.14.2.4 ItemType VarSizeElement::type

info and data type
Definition at line 171 of file bsp_exptable.h.
Referenced by bsp_pop_reg(), bsp_push_reg(), bsp_set_tagsize(), deliveryTable_pushPut(), and deliveryTable_pushSend().
The documentation for this struct was generated from the following file:

- bsp_exptable.h
Chapter 7

BSPonMPI File Documentation

7.1 bsp.c File Reference

Implements the BSPlib primitives.

```
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include "bsp.h"
#include "bsp_memreg.h"
#include "bsp_mesgqueue.h"
#include "bsp_delivtable.h"
#include "bsp_reqtable.h"
#include "bsp_private.h"
#include "bsp_alloc.h"
#include "bsp_abort.h"
#include <config.h>
```

Include dependency graph for bsp.c:
Initialisation

- void **bsp_init** (void(*spmd_part)(void), int argc, char **argv[])  
  *Initializes BSPonMPI.*

- void **bsp_begin** (int maxprocs)  
  *Marks the start of the SPMD code.*

- void **bsp_end** ()  
  *Ends the SPMD code.*

Halt

- void **bsp_abort** (const char *format,...)
Aborts the program printing a message to the standard error output.

**Enquiry**

- `int bsp_nprocs ()`
  Returns the number of processors available / allocated.

- `int bsp_pid ()`
  Returns the rank of the processor.

- `double bsp_time ()`
  The time in seconds since `bsp_begin()`.

**Superstep**

- `void bsp_sync ()`
  Separates two supersteps.

**DRMA**

- `void bsp_push_reg (const void *ident, int size)`
  Makes the memory location with specified size available for DRMA operations at the next and additional supersteps.

- `void bsp_pop_reg (const void *ident)`
  Deregisters the memory location.

- `void bsp_put (int pid, const void *src, void *dst, int offset, int nbytes)`
  Puts a block of data in the memory of some other processor at the next superstep.

- `void bsp_get (int pid, const void *src, int offset, void *dst, int nbytes)`
  Gets a block of data from the memory of some other processor at the next superstep.

**BSMP**

- `void bsp_send (int pid, const void *tag, const void *payload, int payload_nbytes)`
  Sends message to a processor.

- `void bsp_qsize (int *nmessages, int *accum_nbytes)`
  Gives the number of messages and the sum of the payload sizes in queue.
• void bsp_get_tag (int *restrict status, void *restrict tag)
  Retrieves the tag and payload size of the current message in queue.

• void bsp_move (void *payload, int reception_nbytes)
  Dequeue the current message.

• void bsp_set_tagsize (int *tag_nbytes)
  Sets the tag size at the next superstep.

High Performance

• void bsp_hpput (int pid, const void *src, void *dst, int offset, int nbytes)
  Puts a block of data in the memory of some other processor at the next superstep.

• void bsp_hpget (int pid, const void *src, int offset, void *dst, int nbytes)
  Gets a block of data from the memory of some other processor at the next superstep.

• int bsp_hpmove (const void **tag_ptr, const void **payload_ptr)
  Dequeue the current message in an unbuffered way.

Defines

• #define DELIVTAB_SIZE 1000
• #define REQTAB_SIZE 1000
• #define MEMREG_SIZE 100
• #define MESGQ_SIZE 1000

7.1.1 Detailed Description

Implements the BSPlib primitives.

Author:
  Wijnand Suijlen

Definition in file bsp.c.

7.1.2 Define Documentation

7.1.2.1 #define DELIVTAB_SIZE 1000

Definition at line 370 of file bsp.c.
Referenced by bsp_begin().
7.1 bsp.c File Reference

7.1.2.2 #define MEMREG_SIZE 100

Definition at line 372 of file bsp.c.
Referenced by bsp_begin().

7.1.2.3 #define MESGQ_SIZE 1000

Definition at line 373 of file bsp.c.
Referenced by bsp_begin().

7.1.2.4 #define REQTAB_SIZE 1000

Definition at line 371 of file bsp.c.
Referenced by bsp_begin().

7.1.3 Function Documentation

7.1.3.1 void bsp_abort (const char * format, ...)

Aborts the program printing a message to the standard error output.

Parameters:

format uses the same format as printf()

Definition at line 530 of file bsp.c.
References bsp_intern_abort(), and ERR_BSP_ABORT.
Here is the call graph for this function:

7.1.3.2 void bsp_begin (int maxprocs)

Marks the start of the SPMD code.

The code following the call to this function will be executed in parallel with at most
maxprocs processors. The SPMD code must end with a call to bsp_end()

Parameters:

maxprocs Denotes the requested number of processors. The actual allocated num-
ber of processors may be less. Calling

bsp_begin(bsp_nprocs())

allocates the maximum number of processors.
See also:

bsp_nprocs()
bsp_end()

Definition at line 445 of file bsp.c.
References _BSPObject::begintime, bsp, bsp_free(), bsp_malloc, _-BSPObject::communicator, _-BSPObject::delivery_received_table, _-BSPObject::delivery_table, deliveryTable_initialize(), DELIVTAB_SIZE, _-BSPObject::memory_register, memoryRegister_initialize(), MEMREG_SIZE, MESGQ_SIZE, _BSPObject::message_queue, messageQueue_initialize(), MIN, _BSPObject::nprocs, _BSPObject::rank, REQTAB_SIZE, _BSPObject::request_-received_table, _BSPObject::request_table, and requestTable_initialize().

Here is the call graph for this function:

7.1.3.3 void bsp_end ()

Ends the SPMD code.

This function must be called after all SPMD code has been executed. The code after this call is executed by processor 0 only.

Definition at line 507 of file bsp.c.
References bsp, _BSPObject::delivery_received_table, _BSPObject::delivery_table, deliveryTable_destruct(), _BSPObject::memory_register, memoryRegister_destruct(), _BSPObject::message_queue, messageQueue_destruct(), _BSPObject::request_-received_table, _BSPObject::request_table, and requestTable_destruct().

Here is the call graph for this function:
7.1.3.4 void bsp_get (int pid, const void * src, int offset, void * dst, int nbytes)

Gets a block of data from the memory of some other processor at the next superstep.

This function is buffered, i.e.: The data is retrieved from thedestination processor at
the start of the next bsp_sync(). Translation of the src pointer is performed with help
of earlier calls to bsp_push_reg()

Parameters:

    pid      Ranks of the source (remote) processor
    src      Pointer to source location using a pointer to a local memory region
    offset   offset from src in bytes
    dst      Pointer to destination location
    nbytes   Number of bytes to be received

See also:

    bsp_push_reg()

Definition at line 678 of file bsp.c.

References bsp, ReqElement::dst, _BSPObject::memory_register, memoryRegister_-memoized_find(), ReqElement::offset, _BSPObject::request_table, requestTable_-push(), ReqElement::size, and ReqElement::src.

Referenced by bsp_hppget().

Here is the call graph for this function:

7.1.3.5 void bsp_get_tag (int * restrict status, void * restrict tag)

Retrieves the tag and payload size of the current message in queue.

Parameters:

    status  the size of the payload of the current message, or when the queue is empty
      -1
    tag      a pointer to a memory location big enough to contain a tag.

Definition at line 731 of file bsp.c.

References bsp, _BSPObject::message_queue, messageQueue_getNumber(), and messageQueue_getTag().

Here is the call graph for this function:
7.1.3.6  void bsp_hpget (int pid, const void * src, int offset, void * dst, int nbytes)

Gets a block of data from the memory of some other processor at the next superstep. This should be the high performance unbuffered version of bsp_get(). However currently it is the same as bsp_get()

Parameters:
  - **pid**  Ranks of the source (remote) processor
  - **src**  Pointer to source location using a pointer to a local memory region
  - **offset**  offset from src in bytes
  - **dst**  Pointer to destination location
  - **nbytes**  Number of bytes to be received

See also:
  - bsp_push_reg()

Definition at line 804 of file bsp.c.

References bsp_get().

Here is the call graph for this function:

7.1.3.7  int bsp_hpmove (const void ** tag_ptr, const void ** payload_ptr)

Dequeue the current message in an unbuffered way.

Parameters:
  - **tag_ptr**  a pointer to reference of memory location which will contain the tag
  - **payload_ptr**  a pointer to a reference of a memory location which will contain the payload

Returns:
  - the payload size of the dequeued message
7.1 bsp.c File Reference

Definition at line 817 of file bsp.c.
References bsp, _BSPObj::message_queue, messageQueue_getNumber(), and messageQueue_hpdequeue().

Here is the call graph for this function:

7.1.3.8  void bsp_hpput (int pid, const void * src, void * dst, int offset, int nbytes)

Puts a block of data in the memory of some other processor at then next superstep.
This should be the high performance unbuffered version of bsp_put, but currently these two functions are the same

Parameters:
  pid  rank of destination (remote) processor
  src  pointer to source location on source (local) processor
  dst  pointer to destination location on source processor. Translation of addresses is performed with help of earlier calls to bsp_push_reg()
  offset  offset from dst in bytes (comes in handy when working with arrays)
  nbytes  number of bytes to be copied

See also:
  bsp_put()

Definition at line 787 of file bsp.c.
References bsp_put().
Here is the call graph for this function:

7.1.3.9  void bsp_init (void(*)(void) spmd_part, int argc, char * argv[])

Initializes BSPonMPI.

Bug
Because several MPI implementations require that command line arguments are supplied, a call to this function at the start of the program is mandatory
Example

```c
void do_something()
{
    bsp_begin(bsp_nprocs())
    ... a parallel program ...
    bsp_end()
}

int main(int argc, char *argv[])
{
    bsp_init( &do_something, argc, argv);
    ... optional sequential code ...
    do_something();
    ... optional sequential code ...
}
```

Parameters:
- `spmd_part` reference to the SPMD function
- `argc` obtained from function `main()`
- `argv` obtained from function `main()`

See also:
- `bsp_begin()`
- `bsp_end()`
- `bsp_nprocs()`

Definition at line 415 of file `bsp.c`.

References `bsp`, `_BSPObject::nprocs`, and `_BSPObject::rank`.

### 7.1.3.10 void bsp_move (void *payload, int reception_nbytes)

Dequeue the current message.

Parameters:
- `payload` A pointer to a memory location big enough to contain the payload or `reception_nbytes`
- `reception_nbytes` The maximum number of bytes to copy

Definition at line 746 of file `bsp.c`.

References `bsp`, `_BSPObj ect::message_queue`, and messageQueue_dequeue().

Here is the call graph for this function:

```
+-------------------+-----------------+-----------------+-------------------+
| bsp_move          | messageQueue_dequeue | noslots          |
+-------------------+-----------------+-----------------+-------------------+
```

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
7.1.3.11 int bsp_nprocs ()

Returns the number of processors available / allocated.

Returns:
• if bsp_init() is not yet called: -1 . Note that it is allowed to call
  bsp_begin(bsp_nprocs())
• if bsp_init() is called, but not bsp_begin(): number of processors available
• if bsp_init() and bsp_begin() are called: number of processors allocated

Definition at line 552 of file bsp.c.
References bsp, and _BSPObject::nprocs.

7.1.3.12 int bsp_pid ()

Returns the rank of the processor.

Returns:
The rank of the processor

Definition at line 563 of file bsp.c.
References bsp, and _BSPObject::rank.

7.1.3.13 void bsp_pop_reg (const void * ident)

Deregisters the memory location.

Parameters:
ident pointer to memory location

See also:
bsp_push_reg()

Definition at line 633 of file bsp.c.
References PopRegObject::address, bsp, VarSizeElement::data, __BSPObject::delivery_table, deliveryTable_push(), VarSizeElement::info, __Var-SzInfo::pop, popreg, __BSPObject::rank, VarSizeElement::size, and VarSize-Element::type.

Here is the call graph for this function:

```plaintext
bsp_pop_reg deliveryTable_push
```

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
7.1.3.14  void bsp_push_reg (const void *ident, int size)

Makes the memory location with specified size available for DRMA operations at the
next and additional supersteps.

Parameters:
  ident  pointer to memory location
  size   of memory block

Note:
  In this version of BSPonMPI the parameter size is ignored

See also:
  bsp_pop_reg()

Definition at line 616 of file bsp.c.
References PushRegObject::address, bsp, VarSizeElement::data, _-
BSPObj ect::delivery_table, deliveryTable_push(), VarSizeElement::info, _-
BSPObj ect::nprocs, _VarSzInfo::push, pushreg, VarSizeElement::size, and Var-
SizeElement::type.

Here is the call graph for this function:

7.1.3.15  void bsp_put (int pid, const void *src, void *dst, int offset, int nbytes)

Puts a block of data in the memory of some other processor at the next superstep.
This function is buffered, i.e.: the contents of src is copied to a buffer and transmitted
at the next bsp_sync()

Parameters:
  pid   rank of destination (remote) processor
  src   pointer to source location on source (local) processor
  dst   pointer to destination location on source processor. Translation of addresses is
       performed with help of earlier calls to bsp_push_reg()
  offset offset from dst in bytes (comes in handy when working with arrays)
  nbytes number of bytes to be copied

See also:
  bsp_push_reg()

Definition at line 655 of file bsp.c.
References bsp, _BSPObj ect::delivery_table, deliveryTable_pushPut(), _BSPObj ect::memory_register, and memoryRegister_memoized_find().

Referenced by bsp_hpput().

Here is the call graph for this function:

7.1.3.16  void bsp_qsize (int *restrict nmessages, int *restrict accum_nbytes)

Gives the number of messages and the sum of the payload sizes in queue.

Parameters:

$nmessages$ pointer to an int. The value of this integer will be set to the number of messages in queue

$accum_nbytes$ pointer to an int. The value of this integer will be set to the sum of payload sizes in all messages.

Definition at line 717 of file bsp.c.

References bsp, _BSPObj ect::message_queue, messageQueue_getAccumSize(), and messageQueue_getNumber().

Here is the call graph for this function:

7.1.3.17  void bsp_send (int pid, const void *tag, const void *payload, int payload_nbytes)

Sends message to a processor.

You may supply a tag and a payload. The default size of the tag is 0. To change the tag size you can use bsp_set_tagsize().

Parameters:

$pid$  Rank of the destination processor

tag  pointer to the tag

$payload$  pointer to the payload

$payload_nbytes$  size of the payload
Definition at line 704 of file bsp.c.
References bsp, _BSPObject::delivery_table, deliveryTable_pushSend(), _BSPObject::message_queue, and messageQueue_getTagSize().

Here is the call graph for this function:

7.1.3.18 void bsp_set_tagsize (int * tag_nbytes)

Sets the tag size at the next superstep.

Parameters:

tag_nbytes pointer to an int which should contain the size of the tag in bytes. It becomes current tag size.

Definition at line 756 of file bsp.c.
References bsp, VarSizeElement::data, _BSPObject::delivery_table, deliveryTable_push(), VarSizeElement::info, _BSPObject::message_queue, messageQueue_getTagSize(), _BSPObject::rank, settag, _VarSzInfo::settag, VarSizeElement::size, SetTagObject::tag_size, and VarSizeElement::type.

Here is the call graph for this function:

7.1.3.19 void bsp_sync ()

Seperates two supersteps.

Definition at line 582 of file bsp.c.
References bsp, _BSPObject::communicator, _BSPObject::delivery_received_table, _BSPObject::delivery_table, deliveryTable_execute(), deliveryTable_reset(), expandableTable_comm(), expandableTable_reset(), _BSPObject::memory_register, memoryRegister_pack(), _BSPObject::message_queue, messageQueue_reset(), _BSPObject::request_received_table, _BSPObject::request_table, and requestTable_execute().

Here is the call graph for this function:
7.1.3.20 double bsp_time ()

The time in seconds since bsp_begin().

Returns:
The time in seconds since bsp_begin()

Definition at line 572 of file bsp.c.
References _BSObject::begintime, and bsp.
7.2  bsp_abort.c File Reference

implements the internal abort function: bsp_intern_abort()
#include <stdio.h>
#include "bsp_abort.h"
#include <stdlib.h>
#include <mpi.h>

Include dependency graph for bsp_abort.c:

```
Functions

• void bsp_intern_abort (const int err_number, const char *func, const char *file, int line)

  Is used by other library functions when a task could not be successfully completed.
```

7.2.1  Detailed Description

implements the internal abort function: bsp_intern_abort()

Author:
  Wijnand Suijlen

Definition in file bsp_abort.c.

7.2.2  Function Documentation

7.2.2.1  void bsp_intern_abort (const int err_number, const char *func, const char *file, int line)

Is used by other library functions when a task could not be successfully completed.

Parameters:
  err_number one of the Error numbers in ErrorNumbers
  func the function name which caused the bsp_intern_abort
Definition at line 43 of file bsp_abort.c.
References ERR_MESSAGES.
Referenced by bsp_abort(), bspcallocx(), bsp_mallocx(), deliveryTable_execute(), memoryRegister_find(), and memoryRegister_pop().
7.3  bsp_abort.h File Reference

Defines the prototype of an ‘abort’ function for internal usage: bsp_intern_abort().

This graph shows which files directly or indirectly include this file:

---

ErrorNumbers

Error number definitions and their translation

- #define ERR_GET_DELIVERED 5
  When a ‘bsp_get’ gets delivered.

- #define ERR_STACK_COUNTER_OVERFLOW 4
  used in MemoryRegister when the stack counter becomes too big

- #define ERR_POP_REG_WITHOUT_PUSH 3
  used in MemoryRegister when an item could not be found

- #define ERR_NOT_ENOUGH_MEMORY 2
  when memory could not be allocated

- #define ERR_BSP_ABORT 1
  when the user calls bsp_abort()

- #define ERR_MESSAGES
7.3 bsp_abort.h File Reference

Translation of error messages.

Functions

- void bsp_intern_abort (const int err_number, const char *func, const char *file, int line)

  *Is used by other library functions when a task could not be successfully completed.*

7.3.1 Detailed Description

Defines the prototype of an ‘abort’ function for internal usage: bsp_intern_abort().

Author:

Wijnand Suijlen

Definition in file bsp_abort.h.

7.3.2 Define Documentation

7.3.2.1 #define ERR_BSP_ABORT 1

when the user calls bsp_abort()

Definition at line 50 of file bsp_abort.h.

Referenced by bsp_abort().

7.3.2.2 #define ERR_GET_DELIVERED 5

When a ‘bsp_get’ gets delivered.

< this is impossible

Definition at line 42 of file bsp_abort.h.

Referenced by deliveryTable_execute().

7.3.2.3 #define ERR_MESSAGES

Value:

```
{ "abnormal program termination\n", "Not enough memory\n", "bsp_pop_reg without bsp_push_reg\n", "stack counter overflow\n", "a bsp_get() is delivered! contact the library maintainer\n" }
```
Translation of error messages.
Definition at line 53 of file bsp_abort.h.
Referenced by bsp_intern_abort().

7.3.2.4  #define ERR_NOT_ENOUGH_MEMORY 2
when memory could no be allocated
Definition at line 48 of file bsp_abort.h.
Referenced by bspcallocx(), and bsp_mallocx().

7.3.2.5  #define ERR_POP_REG_WITHOUT_PUSH 3
used in MemoryRegister when an item could not be found
Definition at line 46 of file bsp_abort.h.
Referenced by memoryRegister_find(), and memoryRegister_pop().

7.3.2.6  #define ERR_STACK_COUNTER_OVERFLOW 4
used in MemoryRegister when the stack counter becomes to big
Definition at line 44 of file bsp_abort.h.

7.3.3  Function Documentation

7.3.3.1  void bsp_intern_abort (const int err_number, const char *func, const
        char *file, int line)
Is used by other library functions when a task could not be successfully completed.

Parameters:
  err_number  one of the Error numbers in ErrorNumbers
  func  the function name which caused the bsp_intern_abort
  file  the file name
  line  the line number

Definition at line 43 of file bsp_abort.c.
References ERR_MESSAGES.
Referenced by bsp_abort(), bspcallocx(), bsp_mallocx(), deliveryTable_execute(),
memoryRegister_find(), and memoryRegister_pop().
7.4 bsp_alloc.h File Reference

provides wrapper functions for malloc(), calloc() and free().
#include <config.h>
#include <stdlib.h>
#include "bsp_abort.h"

Include dependency graph for bsp_alloc.h:

This graph shows which files directly or indirectly include this file:

Defines

• #define bsp_malloc(n, sz)

wrapper macro for bsp_mallocx()

• #define bsp_calloc(n, sz)

wrapper macro bsp_callocx()
Functions

- static void * bsp_mallocx (const size_t n, const size_t sz, const char *func, const char *file, int line)
  
  allocates memory.

- static void * bsp_callocx (const size_t n, const size_t sz, const char *func, const char *file, int line)
  
  allocates cleared (zeroed) memory.

- static void bsp_free (void *ptr)
  
  frees memory.

7.4.1 Detailed Description

provides wrapper functions for malloc(), calloc() and free().

Author:

Wijnand Suijlen

Definition in file bsp_alloc.h.

7.4.2 Define Documentation

7.4.2.1 #define bsp_calloc(n, sz)

Value:

bsp_callocx((n), (sz), \ __func__, __FILE__, __LINE__ )

wrapper macro bsp_callocx()

Definition at line 44 of file bsp_alloc.h.

Referenced by deliveryTable_initialize(), expandableTable_initialize(), memory-Register_expand(), memoryRegister_initialize(), and newMemRegInfoAtPush().

7.4.2.2 #define bsp_malloc(n, sz)

Value:

bsp_mallocx((n), (sz), \ __func__, __FILE__, __LINE__ )
wrapper macro for bsp_malloctx()

Definition at line 40 of file bsp_alloc.h.

Referenced by bsp_begin(), expandableTable_expand(), and expandableTable__initialize().

### 7.4.3 Function Documentation

#### 7.4.3.1 static void bsp_callocx (const size_t n, const size_t sz, const char * func, const char * file, int line) [inline, static]

allocates cleared (zeroed) memory.

**Parameters:**
- **n** number of elements to be allocated
- **sz** size of one element
- **func** Function from where this function is called
- **file** File from where this function is called
- **line** line number from where this function is called

**Returns:**
- a pointer to a memory location big enough to contain the data or NULL when a region of size 0 is requested

Definition at line 83 of file bsp_alloc.h.

References bsp_intern_abort(), and ERR_NOT_ENOUGH_MEMORY.

Here is the call graph for this function:

```
bsp_callocx bsp_intern_abort
```

#### 7.4.3.2 static void bsp_free (void * ptr) [inline, static]

frees memory.

**Parameters:**
- **ptr** A pointer created by a bsp_malloc() or a bsp_calloc()

Definition at line 102 of file bsp_alloc.h.

Referenced by bsp_begin(), expandableTable_destruct(), expandableTable_expant(), memoryRegister_destruct(), memoryRegister_expand(), and newMemRegInfoAtPush().
7.4.3.3 static void ∗ bsp_mallocx (const size_t n, const size_t sz, const char ∗ func, const char ∗ file, int line) [inline, static] allocates memory.

Note: Use the macro’s bsp_malloc() and bsp_calloc() instead

Parameters:

- **n** number of elements to be allocated
- **sz** size of one element
- **func** Function from where this function is called
- **file** File from where this function is called
- **line** line number from where this function is called

Returns:

- a pointer to a memory location big enough to contain the data or NULL when a region of size 0 is requested

Definition at line 58 of file bsp_alloc.h.

References bsp_intern_abort(), and ERR_NOT_ENOUGH_MEMORY.

Here is the call graph for this function:
7.5 bsp_delivtable.c File Reference

Implements the methods on a DeliveryTable object.

```
#include "bsp_exptable.h"
#include "bsp_memreg.h"
#include "bsp_mesgqueue.h"
```

Include dependency graph for bsp_delivtable.c:

```
stdlib.h
memory.h
bsp_alloc.h
config.h
mpi.h
bsp_abort.h
```

Functions

- static void `exec_put` (const `VarSizeElement` element, const char *restrict offset)
- void `deliveryTable_execute` (`ExpandableTable` *restrict table, `ExpandableTable` *restrict memreg, `ExpandableTable` *restrict mesgq)

```
Executes a DeliveryTable object, i.e.
```

7.5.1 Detailed Description

Implements the methods on a DeliveryTable object.

**Author:**

Wijnand Suijlen

Definition in file `bsp_delivtable.c`.

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
7.5.2 Function Documentation

7.5.2.1 void deliveryTable_execute (ExpandableTable *restrict table, ExpandableTable *restrict memreg, ExpandableTable *restrict mesgq)

Executes a DeliveryTable object, i.e. performs all the actions to be taken when a DeliveryTable is received

Parameters:
- **table** Reference to a DeliveryTable
- **memreg** Reference to a MemoryRegister
- **mesgq** Reference to a MessageQueue

Definition at line 61 of file bsp_delivtable.c.

References bsp_intern_abort(), ERR_GET_DELIVERED, exec_put(), get, memoryRegister_pop(), memoryRegister_push(), messageQueue_push(), messageQueue_setTagSize(), popreg, pushreg, put, send, and settag.

Referenced by bsp_sync().

Here is the call graph for this function:

```
7.5.2.2 static void exec_put (const VarSizeElement element, const char *restrict offset) [inline, static]

Definition at line 38 of file bsp_delivtable.c.

References VarSizeElement::info, PutObject::item_count, PutObject::item_size, and _VarSzInfo::put.

Referenced by deliveryTable_execute().
```
7.6 bsp_delivtable.h File Reference

Defines the prototypes of the methods on a DeliveryTable object.

#include "bsp_exptable.h"
#include <config.h>

Include dependency graph for bsp_delivtable.h:

This graph shows which files directly or indirectly include this file:

Functions

- void deliveryTable_execute (ExpandableTable *restrict, ExpandableTable *restrict, ExpandableTable *restrict)

  *initializes a DeliveryTable object*

- static void deliveryTable_initialize (ExpandableTable *restrict table, const int nprocs, const int rows)

  *clears a DeliveryTable object*

- static void deliveryTable_reset (ExpandableTable *restrict table)

  *Frees memory allocated by a DeliveryTable.*

- static void deliveryTable_destruct (ExpandableTable *restrict table)

  *Expands a DeliveryTable.*

- static void delivInfoMake (union SpecInfo *restrict info, int b, int c)

- static void deliveryTable_push (ExpandableTable *restrict table, const int proc, const VarSizeElement elem)
• static void deliveryTable_pushSend (ExpandableTable *restrict table, const int proc, const void *restrict tag, const int tag_size, const void *restrict payload, const int payload_size)

        Adds a 'bsp_send()' to a DeliveryTable.

• static void deliveryTable_pushPut (ExpandableTable *restrict table, const int proc, const void *const dst, const void *const restrict data, const int size)

        Adds a bsp_put() to a DeliveryTable.

### 7.6.1 Detailed Description

Defines the prototypes of the methods on a DeliveryTable object.

DeliveryTable provides two functionalities:

1. It is a queue for actions which have to be performed during the next bsp_sync(), e.g. a bsp_set_tagsize() or a bsp_pop_reg()

2. It is a communication buffer for data which has to be sent during the next bsp_sync(), e.g. a bsp_put()

These are the same if you consider sends and puts to be actions as well: A put or send is a memory copy action on the remote processor. Actions like bsp_set_tagsize() which do not need communication are placed in the column of the local processor (the column number is equal to the rank of the local processor). Each action is added to a column as a VarSizeElement. Additionally puts and sends have a block of data appended after the VarSizeElement struct in which one or more bsp_put()’s or bsp_send()’s are combined. An example is shown below.
Note that the size of the entire element is not fixed (hence the name `VarSizeElement`).

**Author:**
Wijnand Suijlen

Definition in file `bsp_delivtable.h`.

### 7.6.2 Function Documentation

#### 7.6.2.1 static void deliveryTable_destruct (ExpandableTable *restrict table)  
[inline, static]

Frees memory allocated by a DeliveryTable.

**Parameters:**
- `table` Reference to a DeliveryTable

Definition at line 89 of file `bsp_delivtable.h`.

References `expandableTable_destruct()`.
Referenced by bsp_end().

Here is the call graph for this function:

```
deliveryTable_destruct  expandableTable_destruct  bsp_free
```

7.6.2.2  
```c
void deliveryTable_execute (ExpandableTable * restrict,
                          ExpandableTable * restrict, ExpandableTable * restrict)
```

7.6.2.3  
```c
static void deliveryTable.expand (ExpandableTable *restrict table,
                                 const int rows)  [inline, static]
```

Expands a DeliveryTable.

Parameters:

- `table`  Reference to a DeliveryTable object
- `rows`  Number of rows which should be added to this table

Definition at line 101 of file bsp_delivtable.h.

References VarSizeElement::data, SpecInfo::deliv, and DelivInfo::latest_pushed_element.

Referenced by deliveryTable_pushPut(), and deliveryTable_pushSend().

7.6.2.4  
```c
static void deliveryTable.initialize (ExpandableTable *restrict table,
                                      const int nprocs, const int rows)  [inline, static]
```

initializes a DeliveryTable object

Parameters:

- `table`  Reference to a DeliveryTable
- `nprocs`  Number of processors to allocate memory for
- `rows`  Number of rows to allocate

Definition at line 67 of file bsp_delivtable.h.

References bsp_calloc, SpecInfo::deliv, DelivInfo::latest_pushed_element, and varElSizeTable_initialize().

Referenced by bsp_begin().

Here is the call graph for this function:

```
deliveryTable_initialize  varElSizeTable_initialize  expandableTable_initialize
```
7.6.2.5 static void deliveryTable_push (ExpandableTable *restrict table, const int proc, const VarSizeElement elem) [inline, static]

Adds a VarSizeElement to a DeliveryTable.

Parameters:
- `table` Reference to a DeliveryTable
- `proc` Destination processor
- `elem` Element which should be added

Definition at line 132 of file bsp_delivtable.h.
Referenced by bsp_pop_reg(), bsp_push_reg(), and bsp_set_tagsize().

7.6.2.6 static void deliveryTable_pushPut (ExpandableTable *restrict table, const int proc, const void *const dst, const void *const restrict data, const int size) [inline, static]

Adds a `bsp_put()` to a DeliveryTable.

Parameters:
- `table` Reference to a DeliveryTable
- `proc` Destination processor
- `dst` Destination pointer (remote)
- `data` Source pointer (local)
- `size` Size of the data

If the last element was also a `bsp_put()` then do message combining

Definition at line 261 of file bsp_delivtable.h.
References VarSizeElement::data, deliveryTable_expand(), VarSizeElement::info, PutObject::item_count, PutObject::item_size, MAX, no_slots(), _VarSzInfo::put, put, VarSizeElement::size, and VarSizeElement::type.
Referenced by bsp_put(), and requestTable_execute().
Here is the call graph for this function:

7.6.2.7 static void deliveryTable_pushSend (ExpandableTable *restrict table, const int proc, const void *restrict tag, const int tag_size, const void *restrict payload, const int payload_size) [inline, static]

Adds a `bsp_send()` to a DeliveryTable.

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
7.6.2.8 static void deliveryTable_reset (ExpandableTable *restrict table)  
    [inline, static]

clears a DeliveryTable object

Parameters:
    table  Reference to a DeliveryTable

Definition at line 79 of file bsp_delivtable.h.
References expandableTable_reset().
Referenced by bsp_sync().
Here is the call graph for this function:

7.6.2.9 static void delivInfoMake (union SpecInfo *restrict info, int b, int c)  
    [static]

Definition at line 122 of file bsp_delivtable.h.
7.7 bsp_exptable.c File Reference

Implements the communication method on an ExpandableTable.

```c
#include "bsp_exptable.h"
#include "bsp_alloc.h"
#include "bsp_abort.h"
```

Include dependency graph for bsp_exptable.c:

![Dependency Graph]

Functions

- void **expandableTable_comm**(const **ExpandableTable** ∗restrict **send**, **ExpandableTable** ∗restrict **recv**, **MPI_Comm** **communicator**)

  *communicates the table to the other processor.*

7.7.1 Detailed Description

Implements the communication method on an ExpandableTable.

**Author:**

Wijnand Suijlen

Definition in file **bsp_exptable.c**.

7.7.2 Function Documentation

7.7.2.1 void **expandableTable_comm**(const **ExpandableTable** ∗restrict **send**, **ExpandableTable** ∗restrict **recv**, **MPI_Comm** **communicator**)

  *communicates the table to the other processor.*

  Each column of the table corresponds to a chunk of data which is to be send to the processor with rank equal to the column number. The received data is equally ordered,
i.e.: each column corresponds to the data received from the processor with rank equal to the column number.

**Parameters:**

- **send** Reference to a table which should be send to the other processors
- **recv** Reference to a table which could contain the data received. This table is expanded if necessary
- **communicator** MPI Communicator group which exchange the tables.

Definition at line 47 of file bsp_exptable.c.

Referenced by bsp_sync().
7.8 bsp_exptable.h File Reference

Defines struct’s, prototypes and inlined functions of the ExpandableTable, FixedElSizeTable and VarElSizeTable ‘classes’ and some struct’s of DeliveryTable, RequestTable, MemoryRegister and MessageQueue.

```c
#include <stdlib.h>
#include <memory.h>
#include "bsp_alloc.h"
#include <mpi.h>
#include <config.h>
```

Include dependency graph for bsp_exptable.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- **struct** `MesgQInfo`
  
  *Additional data needed by a MessageQueue object.*

- **struct** `MemRegInfo`
  
  *Additional data needed by a MemoryRegister object.*
• struct ReqElement
  
  Data element stored in a RequestTable object.

• struct PutObject
  
  additional info for a bsp_put()

• struct SendObject
  
  additional info for a bsp_send()

• struct PushRegObject
  
  Additional info for a bsp_push_reg().

• struct PopRegObject
  
  Additional info for a bsp_pop_Reg().

• struct SetTagObject
  
  Additional info for a bsp_set_tagsize().

• union _VarSzInfo
  
  Data structure for additional info in a VarSizeElement.

• struct VarSizeElement
  
  Data element stored in a VarElSizeTable (e.g.

• struct DelivInfo
  
  Additional data for a DeliveryTable.

• union SpecInfo
  
  Datastructure storing specific info.

• struct _ExpandableTable
  
  a table with nprocs number columns and rows number of rows, where each row has a height of slot_size bytes.

MemoryRegister data structures

• typedef char * MemRegElement
  
  Data element stored in a MemoryRegister.

VarElSizeTable data structures

• typedef enum _ItemType ItemType
• enum _ItemType {
    popreg, pushreg, put, get,
    send, settag
  }
ExpandableTable

- typedef _ExpandableTable ExpandableTable
  
a table with nprocs number columns and rows number of rows, where each row has a
  height of slot_size bytes.

- void expandableTable_comm (const ExpandableTable *restrict send,
  ExpandableTable *restrict recv, MPI_Comm communicator)
  
  communicates the table to the other processor.

- static void expandableTable_initialize (ExpandableTable *restrict table, const int
  nprocs, const int rows, const int elsize, const union SpecInfo info)

- static void expandableTable_reset (ExpandableTable *restrict table)
  
  clears contents of the table

- static void expandableTable_destruct (ExpandableTable *restrict table)
  
  frees memory taken by an ExpandableTable object

- static void expandableTable_expand (ExpandableTable *restrict table, const int
  rows, const union SpecInfo newinfo)
  
  Add some additional rows to the table.

FixedElSizeTable member functions

A table which contains elements of fixed size. Descendants are MemoryRegister

- static void fixedElSizeTable_initialize (ExpandableTable *restrict table, const int
  nprocs, const int rows, const int elsize, const union SpecInfo info)
  
  initializes an FixedElSizeTable

- static void fixedElSizeTable_push (ExpandableTable *restrict table, const int
  proc, void(*changeinfo)(union SpecInfo *restrict, int, int), const void *restrict
  element)
  
  adds an element for a specific processor to the table and expands the table when
  necessary.

VarElSizeTable member functions

A table with variable-length elements

- static void varElSizeTable_initialize (ExpandableTable *restrict table, const int
  nprocs, const int rows, const union SpecInfo info)
  
  Initializes a VarElSizeTable.
• static int varElSizeTable_push (ExpandableTable *restrict table, const int proc, void(*changeinfo)(union SpecInfo *restrict, int, int), VarSizeElement elem)

  Adds an element to the table and expands the table when necessary.

Defines

• #define MAX(x, y) ( ((x) > (y))?(x):(y))
  gives the maximum of x and y

• #define MIN(x, y) ( ((x) < (y))?(x):(y))
  gives the minimum of x and y

Functions

• static int no_slots (const int bytes, const int slot_size)
  calculate how many slots are necessary to contain a number of bytes

7.8.1 Detailed Description

Defines struct’s, prototypes and inlined functions of the ExpandableTable, FixedElSizeTable and VarElSizeTable ‘classes’ and some struct’s of DeliveryTable, RequestTable, MemoryRegister and MessageQueue.

See also:
  Implementation in the Architecture description

Author:
  Wijnand Suijlen

Definition in file bsp_exptable.h.

7.8.2 Define Documentation

7.8.2.1 #define MAX(x, y) ( ((x) > (y))?(x):(y))

gives the maximum of x and y

Definition at line 39 of file bsp_exptable.h.

Referenced by deliveryTable_pushPut(), deliveryTable_pushSend(), messageQueue_push(), and varElSizeTable_push().
#define MIN(x, y) ( ((x) < (y))?(x):(y))
gives the minimum of x and y
Definition at line 42 of file bsp_exptable.h.
Referenced by bsp_begin(), and messageQueue_dequeue().

## 7.8.3 Typedef Documentation

### 7.8.3.1 typedef struct _ExpandableTable ExpandableTable

a table with nprocs number columns and rows number of rows, where each row has a height of slot_size bytes.
This table can be communicated to the other processors by means of expandableTable_comm(): each columns is send to processors with rank equal to the column number.

### 7.8.3.2 typedef enum _ItemType ItemType

### 7.8.3.3 typedef char* MemRegElement

Data element stored in a MemoryRegister.
It is a pointer to a registered memory area
Definition at line 88 of file bsp_exptable.h.

## 7.8.4 Enumeration Type Documentation

### 7.8.4.1 enum _ItemType

**Enumerator:**
- popreg
- pushreg
- put
- get
- send
- settag

Definition at line 120 of file bsp_exptable.h.

## 7.8.5 Function Documentation

### 7.8.5.1 void expandableTable_comm (const ExpandableTable *restrict send, ExpandableTable *restrict recv, MPI_Comm communicator)

communicates the table to the other processor.
Each column of the table corresponds to a chunk of data which is to be send to the processor with rank equal to the column number. The received data is equally ordered, i.e.: each column corresponds to the data received from the processor with rank equal to the column number.

**Parameters:**
- **send** Reference to a table which should be send to the other processors
- **recv** Reference to a table which could contain the data received. This table is expanded if necessary
- **communicator** MPI Communicator group which exchange the tables.

Definition at line 47 of file bsp_exptable.c.
Referenced by bsp_sync().

### 7.8.5.2 static void expandableTable_destruct (ExpandableTable *restrict table) [inline, static]

frees memory taken by an ExpandableTable object

**Parameters:**
- **table** Reference to an ExpandableTable object

Definition at line 264 of file bsp_exptable.h.
References bsp_free().
Referenced by deliveryTable_destruct(), memoryRegister_destruct(), messageQueue_destruct(), and requestTable_destruct().
Here is the call graph for this function:

```
expandableTable_destruct -> bsp_free
```

### 7.8.5.3 static void expandableTable_expand (ExpandableTable *restrict table, const int rows, const union SpecInfo newinfo) [inline, static]

Add some additional rows to the table.

**Parameters:**
- **table** Reference to an ExpandableTable object
- **rows** Number of rows to add
- **newinfo** The object specific information may have to be changed. This can be supplied via this parameter

Definition at line 278 of file bsp_exptable.h.
References bsp_free(), and bsp_malloc.

Referenced by fixedElSizeTable_push(), memoryRegister_expand(), messageQueue_-expand(), requestTable_expand(), and varElSizeTable_push().

Here is the call graph for this function:

![Call Graph]

### 7.8.5.4 static void expandableTable_initialize (ExpandableTable ∗restrict table, const int nprocs, const int rows, const int elsize, const union SpecInfo info) [inline, static]

Definition at line 236 of file bsp_exptable.h.

References bsp_calloc, and bsp_malloc.

Referenced by fixedElSizeTable_initialize(), and varElSizeTable_initialize().

### 7.8.5.5 static void expandableTable_reset (ExpandableTable ∗restrict table)

[inline, static]

clears contents of the table

**Parameters:**

- **table**  Reference to an ExpandableTable object

Definition at line 254 of file bsp_exptable.h.

Referenced by bsp_sync(), deliveryTable_reset(), and messageQueue_reset().

### 7.8.5.6 static void fixedElSizeTable_initialize (ExpandableTable ∗restrict table, const int nprocs, const int rows, const int elsize, const union SpecInfo info) [inline, static]

initializes an FixedElSizeTable

**Parameters:**

- **table**  Reference to an ExpandableTable
- **nprocs**  Number of processors
- **rows**  Number of rows
- **elsize**  Size of an element
- **info**  addation info

Definition at line 310 of file bsp_exptable.h.

References expandableTable_initialize().
Referenced by memoryRegister_initialize(), and requestTable_initialize().

Here is the call graph for this function:

```
fixedElSizeTable_initialize -> expandableTable_initialize
```

### 7.8.5.7 static void fixedElSizeTable_push (ExpandableTable *restrict table,
const int proc, void(*)(union SpecInfo *restrict, int, int) changeinfo,
const void *restrict element) [inline, static]

adds an element for a specific processor to the table and expands the table when necessary.

**Parameters:**
- `table` Reference to a FixedSizeElement
- `proc` Destination processor
- `changeinfo` Reference to the function which changes the table info in case of table expansion
- `element` pointer to data element

Definition at line 326 of file bsp_exptable.h.
References expandableTable_expand().
Referenced by memoryRegister_push(), and requestTable_push().

Here is the call graph for this function:

```
fixedElSizeTable_push -> expandableTable_expand -> bsp_free
```

### 7.8.5.8 static int no_slots (const int bytes, const int slot_size) [inline,
static]

calculate how many slots are necessary to contain a number of bytes

**Parameters:**
- `bytes` Number of bytes
- `slot_size` Size of the slots in bytes

Definition at line 60 of file bsp_exptable.h.
References deliveryTable_pushPut(), deliveryTable_pushSend(), messageQueue_dequeue(), messageQueue_hpdequeue(), messageQueue_push(), and varElSizeTable_push().
7.8.5.9 static void varElSizeTable_initialize (ExpandableTable *restrict table, const int nprocs, const int rows, const union SpecInfo info) [inline, static]

Initializes a VarElSizeTable.

**Parameters:**
- **table** Reference to a VarElSizeTable object
- **nprocs** Number of processors
- **rows** Number of rows
- **info** additional info

Definition at line 363 of file bsp_exptable.h.
References expandableTable_initialize().
Referenced by deliveryTable_initialize(), and messageQueue_initialize().
Here is the call graph for this function:

7.8.5.10 static int varElSizeTable_push (ExpandableTable *restrict table, const int proc, void(*)(union SpecInfo *restrict, int, int) changeinfo, VarSizeElement elem) [inline, static]

Adds an element to the table and expands the table when necessary.

**Parameters:**
- **table** Reference to a VarElSizeTable
- **proc** Destination processor
- **changeinfo** Reference to a function which changes table info in case of table expansion
- **elem** Element to be added

Definition at line 377 of file bsp_exptable.h.
References VarSizeElement::data, expandableTable_expand(), MAX, no_slots(), and VarSizeElement::size.
Here is the call graph for this function:
7.9  bsp_memreg.c File Reference

Implements methods on MemoryRegister.

```c
#include "bsp_abort.h"
#include "bsp_memreg.h"
#include "bsp_alloc.h"
```

Include dependency graph for bsp_memreg.c:

```
Functions

• void memoryRegister_initialize (ExpandableTable *restrict table, const int nprocs, const int rows, const int src_proc)
  
  Initializes a MemoryRegister object.

• void memoryRegister_destruct (ExpandableTable *restrict table)
  
  Destructor of MemoryRegister

• void memoryRegister_expand (ExpandableTable *restrict table, const int rows)
  
  Increases the size of a MemoryRegister

• static void newMemRegInfoAtPush (union SpecInfo *restrict info, int rows, int newrows)
  
  Used as parameter of memoryRegister_push.

• void memoryRegister_push (ExpandableTable *restrict table, const int proc, const char *const restrict pointer)
  
  Adds an address of memory location of in a certain processor.
```
• void memoryRegister_pop (ExpandableTable *restrict table, const int proc, const char *const restrict pointer)
  Removed an adress from a MemoryRegister.

• void memoryRegister_pack (ExpandableTable *restrict table)
  Really removes popped elements from the table.

### 7.9.1 Detailed Description

Implements methods on MemoryRegister.

**Author:**

Wijnand Suijlen

Definition in file bsp_memreg.c.

### 7.9.2 Function Documentation

#### 7.9.2.1 void memoryRegister_destruct (ExpandableTable * restrict table)

destructor of MemoryRegister

**Parameters:**

- **table**  Reference to a MemoryRegister object

Definition at line 60 of file bsp_memreg.c.

References bsp_free(), and expandableTable_destruct().

Referenced by bsp_end().

Here is the call graph for this function:

#### 7.9.2.2 void memoryRegister_expand (ExpandableTable * restrict table, const int rows)

increases the size of a MemoryRegister

**Parameters:**

- **table**  Reference to a MemoryRegister object
- **rows**  Number of rows to add

---

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
Definition at line 71 of file bsp_memreg.c.

References bspcalloc, bsp_free(), expandableTable_expand(), MemRegInfo::memoized_data_iter, MemRegInfo::memoized_end, MemRegInfo::memoized_src_proc, MemRegInfo::memoized_srccol, MemRegInfo::numremov, SpecInfo::reg, and MemRegInfo::removed.

Here is the call graph for this function:

```
memoryRegister_expand -> expandableTable_expand -> bsp_free
```

### 7.9.2.3 void memoryRegister_initialize (ExpandableTable *restrict table, const int nprocs, const int rows, const int src_proc)

Initializes a MemoryRegister object.

**Parameters:**
- **table** Reference to a MemoryRegister object
- **nprocs** Number of processors
- **rows** Initial number of rows
- **src_proc** Rank of the local processor

Definition at line 43 of file bsp_memreg.c.

References bspcalloc, fixedElSizeTable_initialize(), MemRegInfo::memoized_data_iter, MemRegInfo::memoized_end, MemRegInfo::memoized_src_proc, MemRegInfo::memoized_srccol, MemRegInfo::numremov, SpecInfo::reg, and MemRegInfo::removed.

Referenced by bsp_begin().

Here is the call graph for this function:

```
memoryRegister_initialize -> fixedElSizeTable_initialize -> expandableTable_initialize
```

### 7.9.2.4 void memoryRegister_pack (ExpandableTable *restrict table)

Really removes popped elements from the table.

Popped elements are marked for removal, but are not actually removed.

**Parameters:**
- **table** Reference to a MemoryRegister

Definition at line 159 of file bsp_memreg.c.

Referenced by bsp_sync().
7.9.2.5  void memoryRegister_pop (ExpandableTable *restrict table, const int proc, const char *const restrict pointer)

Removed an address from a MemoryRegister.

Parameters:
  
  *table Reference to a MemoryRegister  
  *proc Rank of local processor  
  *pointer Local registered address

Definition at line 131 of file bsp_memreg.c. 
References bsp_intern_abort(), and ERR_POP_REG_WITHOUT_PUSH. 
Referenced by deliveryTable_execute(). 
Here is the call graph for this function:

7.9.2.6  void memoryRegister_push (ExpandableTable *restrict table, const int proc, const char *const restrict pointer)

Adds an address of memory location of in a certain processor.

Parameters:
  
  *table Reference to a MemoryRegister object  
  *proc Processor rank of the memory location  
  *pointer Address

Definition at line 117 of file bsp_memreg.c. 
References fixedElSizeTable_push(), and newMemRegInfoAtPush(). 
Referenced by deliveryTable_execute(). 
Here is the call graph for this function:

7.9.2.7  static void newMemRegInfoAtPush (union SpecInfo *restrict info, int rows, int newrows)  [static]

Used as parameter of memoryRegister_push().

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
Parameters:
   info  MemoryRegister info
   rows  Old number of rows
   newrows  New number of rows

Definition at line 96 of file bsp_memreg.c.
References bsp_calloc, and bsp_free().
Referenced by memoryRegister_push().
Here is the call graph for this function:
7.10 bsp_memreg.h File Reference

Defines prototypes of methods on MemoryRegister.

```c
#include <stdlib.h>
#include "bsp_abort.h"
#include "bsp_exptable.h"
#include <config.h>
```

Include dependency graph for bsp_memreg.h:

![Dependency Graph](image)

This graph shows which files directly or indirectly include this file:

```
bsp.c  bsp_memreg.c  bsp_memreg.h
```

Functions

- void `memoryRegister_initialize (ExpandableTable *restrict, const int, const int, const int)`
- void `memoryRegister_destruct (ExpandableTable *restrict)`
- void `memoryRegister_expand (ExpandableTable *restrict, const int)`
- void `memoryRegister_push (ExpandableTable *restrict, const int, const char *const restrict)`
- void `memoryRegister_pop (ExpandableTable *restrict, const int, const char *const restrict)`
- void `memoryRegister_pack (ExpandableTable *restrict)`
- static `MemRegElement memoryRegister_find (const ExpandableTable *restrict table, const int sp, const int dp, const char *const pointer)`
looks up a the address on a remote processor which corresponds to an address on the local processor

- static MemRegElement memoryRegister_memoized_find (const ExpandableTable *restrict table, const int dp, const char *const pointer)
  looks up the address on a remote processor which corresponds to an address on the local processor.

### 7.10.1 Detailed Description

Defines prototypes of methods on MemoryRegister.

MemoryRegister is a table containing memory locations of variables on all processors. Memory locations on the same row are bsp_push_reg()’ed on all processors at the same time (at the same position in the code)

**Author:**
Wijnand Suijlen

Definition in file bsp_memreg.h.

### 7.10.2 Function Documentation

#### 7.10.2.1 void memoryRegister_destruct (ExpandableTable * restrict)

#### 7.10.2.2 void memoryRegister_expand (ExpandableTable * restrict, const int)

#### 7.10.2.3 static MemRegElement memoryRegister_find (const ExpandableTable *restrict table, const int sp, const int dp, const char *const pointer)
  [inline, static]

looks up a the address on a remote processor which corresponds to an address on the local processor

**Parameters:**
- `table` Reference to a MemoryRegister
- `sp` Rank of the local processor
- `dp` Rank of the remote processor
- `pointer` Local address of a registered memory location

**Returns:**
- Remote address

Definition at line 64 of file bsp_memreg.h.

References bsp_intern_abort(), and ERR_POP_REG_WITHOUT_PUSH.
Here is the call graph for this function:

![Call Graph Diagram]

### 7.10.2.4 void memoryRegister_initialize

**Declaration:**

```c
void memoryRegister_initialize (ExpandableTable * restrict, const int, const int, const int)
```

**Description:**

This function initializes the `memoryRegister` with the given parameters.

### 7.10.2.5 static MemRegElement memoryRegister_memoized_find

**Declaration:**

```c
static MemRegElement memoryRegister_memoized_find (const ExpandableTable * restrict table, const int dp, const char * const pointer)
```

**Description:**

This function looks up the address on a remote processor which corresponds to an address on the local processor.

This function is memoized and depends on the MemoryRegisters state of being 'packed'. If not, this function will not function properly.

**Parameters:**

- `table` Reference to a MemoryRegister
- `dp` Rank of the remote processor
- `pointer` Local address of a registered memory location

**Returns:**

Remote address

**Definition:**

Definition at line 91 of file bsp_memreg.h.

Referenced by bsp_get(), and bsp_put().

### 7.10.2.6 void memoryRegister_pack

**Declaration:**

```c
void memoryRegister_pack (ExpandableTable * restrict)
```

### 7.10.2.7 void memoryRegister_pop

**Declaration:**

```c
void memoryRegister_pop (ExpandableTable * restrict, const int, const char * const restrict)
```

### 7.10.2.8 void memoryRegister_push

**Declaration:**

```c
void memoryRegister_push (ExpandableTable * restrict, const int, const char * const restrict)
```
7.11  bsp_mesgqueue.h File Reference

Defines methods on MessageQueue.
#include "bsp_exptable.h"
#include <config.h>
Include dependency graph for bsp_mesgqueue.h:

This graph shows which files directly or indirectly include this file:

Functions

• static void messageQueue_initialize (ExpandableTable *restrict table, const int rows)
  Initializes a MessageQueue.

• static void messageQueue_destruct (ExpandableTable *restrict table)
  Destructor of MessageQueue object.

• static void messageQueue_reset (ExpandableTable *restrict table)
  Clears a MessageQueue.

• static void messageQueue_expand (ExpandableTable *restrict table, const int rows)
  Adds additional rows to a MessageQueue.

• static void messageQueue_push (ExpandableTable *restrict table, char *restrict data, int size)
  Adds a message to the Queue.
7.11 bsp_mesgqueue.h File Reference

- static int messageQueue_setTagSize (ExpandableTable *restrict table, const int tag_size)
  \textit{sets tag size.}

- static int messageQueue_getTagSize (const ExpandableTable *restrict table)
  \textit{Query tag size.}

- static int messageQueue_getNumber (const ExpandableTable *restrict table)
- static int messageQueue_getAccumSize (const ExpandableTable *restrict table)
  \textit{Query the sum of the sizes of all payloads of the messages in queue.}

- static int messageQueue_getTag (ExpandableTable *restrict table, void *restrict tag)
  \textit{Query the tag and payload size of the current message.}

- static int messageQueue_dequeue (ExpandableTable *restrict table, void *restrict payload, int size)
  \textit{Dequeue current message.}

- static int messageQueue_hpdequeue (ExpandableTable *restrict table, const void **const tag, const void **const payload)
  \textit{Dequeues current message.}

### 7.11.1 Detailed Description

Defines methods on MessageQueue.

MessageQueue is queue for holding BSMP messages.

**Author:**
Wijnand Suijlen

Definition in file bsp_mesgqueue.h.

### 7.11.2 Function Documentation

#### 7.11.2.1 static int messageQueue_dequeue (ExpandableTable *restrict table, void *restrict payload, int size) [inline, static]

Dequeue current message.

Copies at least \textit{size} bytes of payload of current message to supplied memory location.

**Parameters:**

- \textit{table} Reference to MessageQueue
payload Reference to a memory location of size \( \text{MIN}(\text{size}, \text{payload size}) \)

size Maximum number of bytes to copy

**Returns:**
actual number of bytes copied

Definition at line 184 of file bsp_mesgqueue.h.
References MIN, and no_slots().
Referenced by bsp_move().
Here is the call graph for this function:

7.11.2.2 static void messageQueue_destruct (ExpandableTable *restrict table)
[inline, static]

Destructor of MessageQueue object.

**Parameters:**

- **table** Reference to MessageQueue object

Definition at line 60 of file bsp_mesgqueue.h.
References expandableTable_destruct().
Referenced by bsp_end().
Here is the call graph for this function:

7.11.2.3 static void messageQueue_expand (ExpandableTable *restrict table, const int rows) [inline, static]

Adds additional rows to a MessageQueue.

**Parameters:**

- **table** Reference to MessageQueue
- **rows** Number of rows to be added

Definition at line 84 of file bsp_mesgqueue.h.
References expandableTable_expand().
Referenced by messageQueue_push().

Here is the call graph for this function:

```
messageQueue_expand expandableTable bsp_free
```

7.11.2.4 static int messageQueue_getAccumSize (const ExpandableTable *restrict table) [inline, static]

Query the sum of the sizes of all payloads of the messages in queue.

**Parameters:**

- `table` Reference to MessageQueue

**Returns:**

The sum of the sizes of all payload of the messages in queue

Definition at line 153 of file bsp_mesgqueue.h.

Referenced by bsp_qsize().

7.11.2.5 static int messageQueue_getNumber (const ExpandableTable *restrict table) [inline, static]

Definition at line 143 of file bsp_mesgqueue.h.

Referenced by bsp_get_tag(), bsp_hpmove(), and bsp_qsize().

7.11.2.6 static int messageQueue_getTag (ExpandableTable *restrict table, void *restrict tag) [inline, static]

Query the tag and payload size of the current message.

**Parameters:**

- `table` Reference to MessageQueue

- `tag` Pointer to memory location which can contain a tag. This memory will be set to the tag

**Returns:**

Payload size

Definition at line 165 of file bsp_mesgqueue.h.

Referenced by bsp_get_tag().
### 7.11.2.7 static int messageQueue_getTagSize (const ExpandableTable *restrict table) [inline, static]

Query tag size.

**Returns:**
- Tag size

Definition at line 133 of file bsp_mesgqueue.h.

Referenced by bsp_send(), and bsp_set_tagsize().

### 7.11.2.8 static int messageQueue_hpdequeue (ExpandableTable *restrict table, const void **const tag, const void **const payload) [inline, static]

Dequeues current message.

Sets the supplied pointers to memory location which contain tag and payload.

**Parameters:**
- `table` Reference to MessageQueue
- `tag` Reference to a pointer which can point to a tag
- `payload` Reference to a pointer which can point to the payload

**Returns:**
- Size of the payload

Definition at line 214 of file bsp_mesgqueue.h.

References no_slots().

Referenced by bsp_hpmove().

Here is the call graph for this function:

```
messageQueue_hpdequeue  no_slots
```

### 7.11.2.9 static void messageQueue_initialize (ExpandableTable *restrict table, const int rows) [inline, static]

Initializes a MessageQueue.

A MessageQueue is a VarElSizeTable with only one column / one processor.

**Parameters:**
- `table` Reference to a MessageQueue object
- `rows` Number of rows

---

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
7.11 bsp_mesgqueue.h File Reference

Definition at line 44 of file bsp_mesgqueue.h.

References MesgQInfo::accum_size, MesgQInfo::inslot_offset, SpecInfo::mesgq, MesgQInfo::n_mesg, MesgQInfo::n_mesg_deq, MesgQInfo::slot_offset, MesgQInfo::tag_size, and varElSizeTable_initialize().

Referenced by bsp_begin().

Here is the call graph for this function:

7.11.2.10 static void messageQueue_push (ExpandableTable ∗ restrict table, char ∗ restrict data, int size) [inline, static]

Adds a message to the Queue.

Because this function is used only in combination with a DeliveryTable the entire combined object is copied to the MessageQueue

Parameters:
  table  Reference to MessageQueue
  data   Reference to a send element in a DeliveryTable
  size   Length of data in bytes

Definition at line 97 of file bsp_mesgqueue.h.

References MAX, messageQueue_expand(), and no_slots().

Referenced by deliveryTable_execute().

Here is the call graph for this function:

7.11.2.11 static void messageQueue_reset (ExpandableTable ∗ restrict table) [inline, static]

Clears a MessageQueue.

Parameters:
  table  Reference to MessageQueue object

Definition at line 69 of file bsp_mesgqueue.h.

References expandableTable_reset().
Referenced by bsp_sync().

Here is the call graph for this function:

```graphviz
messageQueue_reset -> expandableTable_reset
```

7.11.2.12  static int messageQueue_setTagSize (ExpandableTable * restrict table,
        const int tag_size)  [inline, static]

sets tag size.
MessageQueue itself doesn’t use this value. It just remembers it

Parameters:

- **table**  Reference to MessageQueue
- **tag_size**  New tag size

Returns:

Old tag size

Definition at line 122 of file bsp_mesgqueue.h.

Referenced by deliveryTable_execute().
contains a global variable which should not be available to the outside world

#include "bsp_exptable.h"
#include <mpi.h>

Include dependency graph for bsp_private.h:

This graph shows which files directly or indirectly include this file:

Data Structures

• struct _BSPObj ect
  
  global variables used in bsp.c

Typedefs

• typedef _BSPObj ect BSPObj ect

  global variables used in bsp.c

Variables

• static BSPObj ect bsp

  Packed global variables.

7.12.1 Detailed Description

contains a global variable which should not be available to the outside world
Author:
Wijnand Suijlen

Definition in file bsp_private.h.

7.12.2 Typedef Documentation

7.12.2.1 typedef struct _BSPObject BSPObject

global variables used in bsp.c

7.12.3 Variable Documentation

7.12.3.1 BSPObject bsp [static]

Packed global variables.
To keep it private it is not included in bsp.h
Definition at line 69 of file bsp_private.h.

Referenced by bsp_begin(), bsp_end(), bsp_get(), bsp_get_tag(), bsp_hpmove(), bsp_init(), bsp_move(), bsp_nprocs(), bsp_pid(), bsp_pop_reg(), bsp_push_reg(), bsp_put(), bsp_qsize(), bsp_send(), bsp_set_tagsize(), bsp_sync(), and bsp_time().
Implement the 'execute' method on a RequestTable.

```c
#include "bsp_reqtable.h"
#include "bsp_alloc.h"
#include "bsp_delivtable.h"
```

Include dependency graph for bsp_reqtable.c:

```plaintext
bsp_reqtable.c
bsp_reqtable.h
bsp_alloc.h
bsp_delivtable.h
bsp_exptable.h
stdlib.h
config.h
memory.h
mpi.h
bsp_abort.h
```

### Functions

- void `requestTable_execute` (const `ExpandableTable *restrict table`, `ExpandableTable *restrict deliv`)

  Executes the data requests.

#### 7.13.1 Detailed Description

Implement the 'execute' method on a RequestTable.

**Author:**

Wijnand Suijlen

Definition in file `bsp_reqtable.c`.
7.13.2 Function Documentation

7.13.2.1 void requestTable_execute (const ExpandableTable *restrict table,
ExpandableTable *restrict deliv)

Executes the data requests.
Reads all data requests and translates them in data delivery: bsp_put()

Parameters:

- **table**: Reference to RequestTable
- **deliv**: Reference to DeliveryTable

Definition at line 44 of file bsp_reqtable.c.
References deliveryTable_pushPut().
Referenced by bsp_sync().

Here is the call graph for this function:
7.14  bsp_reqtable.h File Reference

defines prototypes and implementations of methods on RequestTable.

```c
#include "bsp_exptable.h"
#include <stdlib.h>
#include <config.h>
```

Include dependency graph for bsp_reqtable.h:

This graph shows which files directly or indirectly include this file:

```
Functions

- void requestTable_execute (const ExpandableTable *restrict, ExpandableTable *restrict)
- static void requestTable_initialize (ExpandableTable *restrict table, const int nprocs, const int rows)
  
  Initializes a RequestTable object.

- static void requestTable_destruct (ExpandableTable *restrict table)
  
  Destructor of RequestTable.

- static void requestTable_expand (ExpandableTable *restrict table, const int rows)
  
  Add additional rows to RequestTable.

- static void newReqInfoAtPush (union SpecInfo *restrict info, int rows, int bla)
- static void requestTable_push (ExpandableTable *restrict table, const int proc, const ReqElement element)

```

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
Adds a data request element to the table.

7.14.1 Detailed Description

defines prototypes and implementations of methods on RequestTable.
RequestTable is a communication buffer for handling data requests (the request part of
a bsp_get()).

Author:
Wijnand Suijlen

Definition in file bsp_reqtable.h.

7.14.2 Function Documentation

7.14.2.1 static void newReqInfoAtPush (union SpecInfo *restrict info, int rows, int bla) [static]

Definition at line 75 of file bsp_reqtable.h.
Referenced by requestTable_push().

7.14.2.2 static void requestTable_destruct (ExpandableTable *restrict table) [inline, static]

Destructor of RequestTable.

Parameters:

\textit{table} Reference to a RequestTable

Definition at line 58 of file bsp_reqtable.h.
References expandableTable_destruct().
Referenced by bsp_end().
Here is the call graph for this function:
7.14.2.3 `void requestTable_execute (const ExpandableTable * restrict, ExpandableTable * restrict)`

Add additional rows to RequestTable.

**Parameters:**
- `table` Reference to a RequestTable
- `rows` Number of rows to add

Definition at line 68 of file `bsp_reqtable.h`.
References `expandableTable_expand()`.

Here is the call graph for this function:

```
requestTable_expand  expandableTable_expand  bsp_free
```

7.14.2.4 `static void requestTable_expand (ExpandableTable * restrict table, const int rows) [inline, static]`

```
Add additional rows to RequestTable.

**Parameters:**
- `table` Reference to a RequestTable
- `rows` Number of rows to add

Definition at line 49 of file `bsp_reqtable.h`.
References `fixedElSizeTable_initialize()`.
Referenced by `bsp_begin()`.

Here is the call graph for this function:

```
requestTable_initialize  fixedElSizeTable_initialize  expandableTable_initialize
```

7.14.2.5 `static void requestTable_initialize (ExpandableTable * restrict table, const int nprocs, const int rows) [inline, static]`

```
Initializes a RequestTable object.

**Parameters:**
- `table` Reference to a RequestTable
- `nprocs` Number of processors
- `rows` Number of rows

Definition at line 49 of file `bsp_reqtable.h`.
References `fixedElSizeTable_initialize()`.
Referenced by `bsp_begin()`.

Here is the call graph for this function:

```
requestTable_initialize  fixedElSizeTable_initialize  expandableTable_initialize
```

7.14.2.6 `static void requestTable_push (ExpandableTable * restrict table, const int proc, const ReqElement element) [inline, static]`

```
Adds a data request element to the table.
```
Parameters:

- **table** Reference to RequestTable
- **proc** Processor rank whereto the request is send
- **element** Description of data request

Definition at line 85 of file bsp_reqtable.h.

References fixedElSizeTable_push(), and newReqInfoAtPush().

Referenced by bsp_get().

Here is the call graph for this function:
8.1 Bug List

**Global bsp_init**  Because several MPI implementations require that command line arguments are supplied, a call to this function at the start of the program is mandatory
Index

_BSOBObject, 19
    begintime, 20
    communicator, 20
    delivery_received_table, 20
    delivery_table, 20
    memory_register, 21
    message_queue, 21
    nprocs, 21
    rank, 21
    request_received_table, 21
    request_table, 21
_ExpandableTable, 23
    _ExpandableTable
        count, 23
        data, 24
        info, 24
        nprocs, 24
        rows, 24
        slot_size, 24
        used_slot_count, 24
_ItemType
    bsp_extable.h, 81
_VarSzInfo, 25
    _VarSzInfo
        pop, 25
        push, 25
        put, 25
        send, 25
        settag, 25
accum_size
    MesqQInfo, 30
address
    PopRegObject, 32
    PushRegObject, 33
begintime
    _BSOObject, 20
bsp
    bsp_private.h, 102
    bsp.c, 43
bsp_abort, 47
    bsp_abort, 47
    bsp_abort.c, 58
    bsp_abort.h, 60
    bsp_abort, 62
    ERR_BSP_ABORT, 61
    ERR_GET_DELIVERED, 61
    ERR_MESSAGES, 61
    ERR_NOT_ENOUGH_MEMORY, 62
    ERR_POP_REG_WITHOUT_-PUSH, 62
    ERR_STACK_COUNTER_-OVERFLOW, 62
bsp_alloc.h, 63
    bsp_calloc, 64
INDEX

bsp_callocx, 65
bsp_free, 65
bsp_malloc, 64
bsp_mallocx, 65
bsp_begin
bsp.c, 47
bsp_calloc
bsp_alloc.h, 64
bsp_callocx
bsp_alloc.h, 65
bsp_delivtable.c, 67
deliveryTable_execute, 68
eexec_put, 68
bsp_delivtable.h, 69
deliveryTable_destruct, 71
deliveryTable_execute, 72
deliveryTable_expand, 72
deliveryTable_initialze, 72
deliveryTable_push, 72
deliveryTable_pushPut, 73
deliveryTable_pushSend, 73
deliveryTable_reset, 74
delivInfoMake, 74
bsp_end
bsp.c, 48
bsp_exptable.c, 75
expandableTable_comm, 75
bsp_exptable.h
get, 81
popreg, 81
pushreg, 81
put, 81
send, 81
settag, 81
bsp_exptable.h, 77
_ItemType, 81
ExpandableTable, 81
expandableTable_comm, 81
expandableTable_destruct, 82
expandableTable_expand, 82
expandableTable_initialize, 83
expandableTable-reset, 83
fixedElSizeTable_initialize, 83
fixedElSizeTable_push, 84
ItemType, 81
MAX, 80
MemRegElement, 81
MIN, 80
no_slots, 84
varElSizeTable_initialize, 84
varElSizeTable_push, 85
bsp_free
bsp_alloc.h, 65
bsp_get
bsp.c, 48
bsp_get_tag
bsp.c, 49
bsp_hpget
bsp.c, 50
bsp_hpmove
bsp.c, 50
bsp_hpput
bsp.c, 51
bsp_init
bsp.c, 51
bsp_intern_abort
bsp_abort.c, 58
bsp_abort.h, 62
bsp_malloc
bsp_alloc.h, 64
bsp_mallocx
bsp_alloc.h, 65
bsp_memreg.c, 86
memoryRegister_destruct, 87
memoryRegister_expand, 87
memoryRegister_initialize, 88
memoryRegister_pack, 88
memoryRegister_pop, 88
memoryRegister_push, 89
newMemRegInfoAtPush, 89
bsp_memreg.h, 91
memoryRegister_destruct, 92
memoryRegister_expand, 92
memoryRegister_find, 92
memoryRegister_initialize, 93
memoryRegister_memoized_find, 93
memoryRegister_pack, 93
memoryRegister_pop, 93
memoryRegister_push, 93
bsp_mesgqueue.h, 94
messageQueue_dequeue, 95
messageQueue_destruct, 96
messageQueue_expand, 96
messageQueue_getAccumSize, 97
messageQueue_getNumber, 97
messageQueue_getTag, 97
messageQueue_getTagSize, 97
messageQueue_hpdequeue, 98
messageQueue_initialze, 98

Generated on Sat Apr 8 12:56:46 2006 for BSPonMPI by Doxygen
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>File/Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>messageQueue_push</td>
<td>bsp.c, 52</td>
</tr>
<tr>
<td>messageQueue_reset</td>
<td>bsp.c, 52</td>
</tr>
<tr>
<td>messageQueue_setTagSize</td>
<td>bsp_private.h, 101</td>
</tr>
<tr>
<td>bsp_move</td>
<td>bsp.c, 52</td>
</tr>
<tr>
<td>bsp_nprocs</td>
<td>bsp.c, 52</td>
</tr>
<tr>
<td>bsp_pid</td>
<td>bsp.c, 53</td>
</tr>
<tr>
<td>bsp_pop_reg</td>
<td>bsp.c, 53</td>
</tr>
<tr>
<td>bsp_private.h</td>
<td>bsp, 102</td>
</tr>
<tr>
<td>BSPObject</td>
<td>bsp_private.h, 102</td>
</tr>
<tr>
<td>bsp_push_reg</td>
<td>bsp.c, 53</td>
</tr>
<tr>
<td>bsp_put</td>
<td>bsp.c, 54</td>
</tr>
<tr>
<td>bsp_qsize</td>
<td>bsp.c, 55</td>
</tr>
<tr>
<td>bsp_reqltable.c</td>
<td>bsp_reqltable.h, 105</td>
</tr>
<tr>
<td>requestTable_execute</td>
<td>bsp_reqltable.h, 104</td>
</tr>
<tr>
<td>newReqInfoAtPush</td>
<td>bsp_reqltable.h, 106</td>
</tr>
<tr>
<td>requestTable_destruct</td>
<td>bsp_reqltable.h, 106</td>
</tr>
<tr>
<td>requestTable_execute</td>
<td>bsp_reqltable.h, 106</td>
</tr>
<tr>
<td>requestTable_expand</td>
<td>bsp_reqltable.h, 107</td>
</tr>
<tr>
<td>requestTable_initialize</td>
<td>bsp_reqltable.h, 107</td>
</tr>
<tr>
<td>requestTable_push</td>
<td>bsp_reqltable.h, 107</td>
</tr>
<tr>
<td>bsp_send</td>
<td>bsp.c, 55</td>
</tr>
<tr>
<td>bsp_set_tagsize</td>
<td>bsp.c, 56</td>
</tr>
<tr>
<td>bsp_sync</td>
<td>bsp.c, 56</td>
</tr>
<tr>
<td>bsp_time</td>
<td>bsp.c, 57</td>
</tr>
<tr>
<td>BSPObject</td>
<td>bsp_private.h, 102</td>
</tr>
<tr>
<td>communicator</td>
<td>_BSPObject, 20</td>
</tr>
<tr>
<td>count</td>
<td>_ExpandableTable, 23</td>
</tr>
<tr>
<td>data</td>
<td>_ExpandableTable, 24</td>
</tr>
<tr>
<td>VarSizeElement</td>
<td>41</td>
</tr>
<tr>
<td>deliv</td>
<td>SpecInfo, 40</td>
</tr>
<tr>
<td>delivery_received_table</td>
<td>_BSPObject, 20</td>
</tr>
<tr>
<td>delivery_table</td>
<td>_BSPObject, 20</td>
</tr>
<tr>
<td>deliveryTable_destruct</td>
<td>bsp_delivtable.h, 71</td>
</tr>
<tr>
<td>deliveryTable_execute</td>
<td>bsp_delivtable.c, 68</td>
</tr>
<tr>
<td>deliveryTable_expand</td>
<td>bsp_delivtable.h, 72</td>
</tr>
<tr>
<td>deliveryTable_initialize</td>
<td>bsp_delivtable.h, 72</td>
</tr>
<tr>
<td>deliveryTable_push</td>
<td>bsp_delivtable.h, 72</td>
</tr>
<tr>
<td>deliveryTable_pushPut</td>
<td>bsp_delivtable.h, 73</td>
</tr>
<tr>
<td>deliveryTable_pushSend</td>
<td>bsp_delivtable.h, 73</td>
</tr>
<tr>
<td>deliveryTable_reset</td>
<td>bsp_delivtable.h, 74</td>
</tr>
<tr>
<td>DelivInfo</td>
<td>27</td>
</tr>
<tr>
<td>DelivInfoMake</td>
<td>latest_pushed_element, 27</td>
</tr>
<tr>
<td>dst</td>
<td>ReqElement, 35</td>
</tr>
<tr>
<td>ERR_BSP_ABORT</td>
<td>bsp_abort.h, 61</td>
</tr>
<tr>
<td>ERR_GET_DELIVERED</td>
<td>bsp_abort.h, 61</td>
</tr>
<tr>
<td>ERR_MESSAGES</td>
<td>bsp_abort.h, 61</td>
</tr>
<tr>
<td>ERR_NOT_ENOUGH_MEMORY</td>
<td>bsp_abort.h, 62</td>
</tr>
<tr>
<td>ERR_POP_REG_WITHOUT_PUSH</td>
<td>bsp_abort.h, 62</td>
</tr>
<tr>
<td>ERR_STACK_COUNTER_-OVERFLOW</td>
<td>bsp_abort.h, 62</td>
</tr>
<tr>
<td>exec_put</td>
<td>bsp_delivtable.c, 68</td>
</tr>
<tr>
<td>ExpandableTable</td>
<td>bsp_exptable.h, 81</td>
</tr>
<tr>
<td>expandableTable_comm</td>
<td>bsp_exptable.c, 75</td>
</tr>
</tbody>
</table>
bsp_exptable.h, 81
expandableTable_destruct
bsp_exptable.h, 82
expandableTable_expand
bsp_exptable.h, 82
expandableTable_initialize
bsp_exptable.h, 83
expandableTable_reset
bsp_exptable.h, 83
fixedElSizeTable_initialize
bsp_exptable.h, 83
fixedElSizeTable_push
bsp_exptable.h, 84
get
bsp_exptable.h, 81
info
_EXPIandableTable, 24
VarSizeElement, 41
inslot_offset
MesgQInfo, 30
item_count
PutObject, 34
SendObject, 37
item_size
PutObject, 34
ItemType
bsp_exptable.h, 81
latest_pushed_element
DelivInfo, 27
MAX
bsp_exptable.h, 80
memoized_data_iter
MemRegInfo, 28
memoized_end
MemRegInfo, 28
memoized_src_proc
MemRegInfo, 28
memoized_srccol
MemRegInfo, 28
memory_register
_BSPObject, 21
memoryRegister_destruct
bsp_memreg.c, 87
bsp_memreg.h, 92
memoryRegister_find
bsp_memreg.h, 92
memoryRegister_initialize
bsp_memreg.c, 88
bsp_memreg.h, 93
memoryRegister_memoized_find
bsp_memreg.h, 93
memoryRegister_pack
bsp_memreg.c, 88
bsp_memreg.h, 93
memoryRegister_pop
bsp_memreg.c, 88
bsp_memreg.h, 93
memoryRegister_push
bsp_memreg.c, 89
bsp_memreg.h, 93
MEMREG_SIZE
bsp.c, 46
MemRegElement
bsp_exptable.h, 81
MemRegInfo, 28
MemRegInfo
memoized_data_iter, 28
memoized_end, 28
memoized_src_proc, 28
memoized_srccol, 28
numremov, 29
removed, 29
mesgq
SpecInfo, 40
MESGQ_SIZE
bsp.c, 47
MesgQInfo, 30
MesgQInfo
accum_size, 30
inslot_offset, 30
n_mesg, 31
n_mesg_deq, 31
slot_offset, 31
tag_size, 31
message_queue
_BSPObject, 21
messageQueue_dequeue
bsp_mesgqueue.h, 95
messageQueue_destruct
bsp_mesgqueue.h, 96
messageQueue_expand
bsp_mesgqueue.h, 96
messageQueue_getAccumSize
bsp_mesgqueue.h, 97
messageQueue_getNumber
bsp_mesgqueue.h, 97
messageQueue_getTag
bsp_mesgqueue.h, 97
messageQueue_getTagSize
bsp_mesgqueue.h, 97
messageQueue_hpdequeue
bsp_mesgqueue.h, 98
messageQueue_initialize
bsp_mesgqueue.h, 98
messageQueue_push
bsp_mesgqueue.h, 99
messageQueue_reset
bsp_mesgqueue.h, 99
messageQueue_setTagSize
bsp_mesgqueue.h, 100
MIN
bsp_exptable.h, 80
n_mesg
   MesgQInfo, 31
n_mesg_deq
   MesgQInfo, 31
newMemRegInfoAtPush
   bsp_memreg.c, 89
newReqInfoAtPush
   bsp_reqtable.h, 106
no_slots
   bsp_exptable.h, 84
nprocs
   _BSPObject, 21
   _ExpandableTable, 24
nuremov
   MemRegInfo, 29
offset
   ReqElement, 35
payload_size
   SendObject, 37
pop
   _VarSzInfo, 25
popreg
   bsp_exptable.h, 81
PushRegObject, 33
PushRegObject
address, 33
put
   _VarSzInfo, 25
   bsp_exptable.h, 81
PutObject, 34
PutObject
   item_count, 34
   item_size, 34
rank
   _BSPObject, 21
reg
   SpecInfo, 40
removed
   MemRegInfo, 29
ReqElement, 35
ReqElement
   dst, 35
   offset, 35
   size, 35
   src, 35
REQTAB_SIZE
   bsp.c, 47
request_received_table
   _BSPObject, 21
request_table
   _BSPObject, 21
requestTable_destruct
   bsp_reqtable.h, 106
requestTable_execute
   bsp_reqtable.c, 104
   bsp_reqtable.h, 106
requestTable_expand
   bsp_reqtable.h, 107
requestTable_initialize
   bsp_reqtable.h, 107
requestTable_push
   bsp_reqtable.h, 107
rows
   _ExpandableTable, 24
send
   _VarSzInfo, 25
   bsp_exptable.h, 81
SendObject, 37
SendObject
item_count, 37
payload_size, 37
tag_size, 37

_settag
    _VarSzInfo, 25
    bsp_exptable.h, 81
SetTagObject, 39
SetTagObject
tag_size, 39

size
    ReqElement, 35
    VarSizeElement, 42
slot_offset
    SpecInfo, 40
    slot_size
        _ExpandableTable, 24
SpecInfo, 40
deliv, 40
mesgq, 40
reg, 40

crc
    ReqElement, 35

tag_size
    SpecInfo, 40
    SendObject, 37
    SetTagObject, 39
type
    VarSizeElement, 42

used_slot_count
    _ExpandableTable, 24

varElSizeTable_initialize
    bsp_exptable.h, 84
varElSizeTable_push
    bsp_exptable.h, 85
VarSizeElement, 41
VarSizeElement
data, 41
info, 41
size, 42
type, 42