Equivalent of memset:

`upc_memset(dst, char, size)`
assign a block of characters to shared memory

Locks

```
// Dynamic lock collectively allocated
{
        upc_lock_t *l;
        l = upc_all_lock_alloc();

        //…
        upc_lock(l);
        // protected section
        upc_unlock(l);

        if( upc_lock_attempt( l ))
          {
                // do something if l currently unlocked
          }

        // unallocates the lock
        if( MYTHREAD == 0)
          upc_lock_free( l );
}

{
        // Dynamic lock globally allocated
        upc_lock_t *l;
        if(MYTHREAD == 3)
          l = upc_global_lock_alloc();
}
```

General utilities

Terminate the UPC program with exit status **`status`** :

High Performance Computing Lab
The George Washington University
801 22nd street NW, Washington DC 20052

UPC QUICK REFERENCE CARD

Keywords

`THREADS`: Total number of threads
`MYTHREAD`: Identification number of the current thread (between **`0`** and **`THREADS-1`**)
`UPC_MAX_BLOCK_SIZE`: Maximum block size allowed by the compilation environment

Shared variable declarations

Shared objects

Shared variables are declared using the type qualifier “shared”. Shared objects must be declared statically (that is, either as global variables or with the keyword static).

Examples of shared object declaration:

```
shared int i;

shared int b[100*THREADS];
```

The following will not compile if you do not specify the number of threads:

```
shared int a[100];
```

All the elements of **`a`** are allocated in thread 0:

```
shared [] int a[100];
```

Distribute the elements in a round robin fashion by chunks of 2 elements: **`a[0]`** and **`a[1]`** are allocated in thread 0; **`a[2]`** and **`a[3]`** in thread 1 ….:

```
shared [2] int a[100];
```

Shared pointers

Pointer to shared object:

```
shared int* p;
```

Shared pointer to shared object:

```
shared int* shared sp;
```

Work sharing

The iteration distribution follows the distribution layout of **`a`**:
`upc_forall(i=0; i<N; i++; &a[i])`

Distributes the iterations in a round-robin fashion with wrapping from the last thread to the first thread:
`upc_forall(i=0; i<N; i++; i)`

Distribute the iterations by consecutive chunks:
`upc_forall(i=0; i<N; i++; i*THREADS/N)`

Synchronization

Memory consistency

These include files set which consistency model, strict or relaxed, is used for the whole program.
`#include "upc_strict.h" or "upc_relaxed.h"`

Sets strict memory consistency for the rest of the file:
`#pragma upc strict`
Sets relaxed memory consistency for the rest of the file:
`#pragma upc relaxed`

All accesses to **`i`** are made using the relaxed consistency model:
`shared relaxed int i;`
All accesses to **`i`** are made using the relaxed consistency model:
`relaxed shared int i;`
All accesses to **`i`** are made using the strict consistency model:
`strict shared int i;`

Synchronize locally the shared memory accesses; it is equivalent to a null strict reference.
`upc_fence;`

Barriers

Globally synchronize the program:
`upc_barrier value;`
`value` is an optional integer.

// Before the barrier
`upc_notify value;`
`value` is an optional integer.

// Non-synchronized statements relative to this on-going barrier
`upc_wait [value];`
`value` is an optional integer.
// After the barrier

Library routines

`upc_threadof(p)` : thread having affinity to the location pointed by **`p`**
`upc_phaseof(p)` : phase associated with the location pointed by **`p`**
`upc_resetphase(p)` : shared address with the phase set to zero pointed by **`p`**
`upc_addrfield(p)` : address field associated with the location pointed by **`p`**
`upc_localsizeof(p)` : size of the local portion pointed by **`p`**
`upc_blocksizeof(p)` : blocking factor associated with object pointed by **`p`**
`upc_elemsizeof(p)` : size of the left-most type of object pointed by **`p`**

Dynamic memory allocation

Three different memory allocation methods are provided by UPC:

`upc_alloc(n)`: allocates at least **`n`** bytes of shared space with affinity to the calling thread . It needs to be called by one thread only.

`upc_global_alloc(n, b)`: globally allocates **`nxb`** bytes of shared data distributed across the threads with a block size of b bytes. It is intended to be called by one thread only.

`upc_all_alloc(n, b)`: collectively allocates **`nxb`** bytes of shared data distributed across the threads with a block size of **`b`** bytes. It is intended to be called by all the threads.

`upc_free(p)`: frees shared memory pointed to by **`p`** from the heap.

String functions in UPC

Equivalent of memcpy :

`upc_memcpy(dst, src, size)`
copy from shared memory to shared memory
`upc_memput(dst, src, size)`
copy from private memory to shared memory
`upc_memget(dst, src, size)`
copy from shared memory to private memory