UPC: Unified Parallel C*

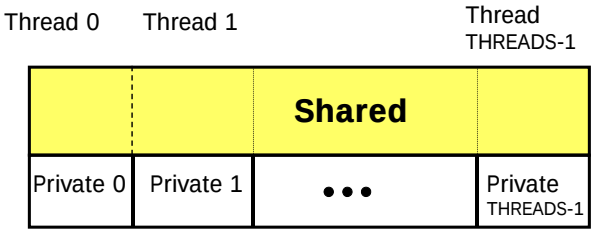
* Slides modified from a presentation prepared by Professor Tarek El-Ghazawi (tarek@gwu.edu)

UPC Execution Model

- Designed for distributed shared memory machines
- Threads working independently in a SPMD fashion
  - MYTHREAD specifies thread index (0..THREADS-1)
  - Number of threads specified at compile-time or run-time
- Allows control of data distribution and work assignment
- Process and Data Synchronization when needed
  - Barriers and split phase barriers
  - Locks and arrays of locks

2

UPC Memory Model



- Shared space with thread affinity, plus private spaces
- A pointer-to-shared can reference all locations in the shared space
- A private pointer may reference only addresses in its private space or addresses in its portion of the shared space
- Static and dynamic memory allocations are supported for both shared and private memory

3

Example: Vector addition

```
//vect_add.c
#define N 100*THREADS
shared int v1[N], v2[N], v1plusv2[N];
void main(){
    int i;
    for(i= MYTHREAD; i<N; i+=THREADS)
        v1plusv2[i]=v1[i]+v2[i];
}
```

Can use

```
upc_forall(i=0; i<N; i++; i)
        v1plusv2[i]=v1[i]+v2[i];
```

Actually translated to

```
for (i=0; i<N; i++)
    if( MYTHREAD == i % THREADS)
        v1plusv2[i]=v1[i]+v2[i];
```

4

The shared qualifier

- Shared array elements and blocks can be spread across the threads
  - shared int x[THREADS] /*One element per thread */
  - shared int y[10][THREADS] /*10 elements per thread */
- Scalar data declarations
- shared int a; /*One copy (affinity to thread 0) */
  int b; /* One private b at each thread */
- Shared data cannot have dynamic scope

5

Examples of data layout

Assume THREADS = 3

```
shared int x;  /*x will have affinity to thread 0 */
shared int y[THREADS];
int z;
shared int A[4][THREADS];
```

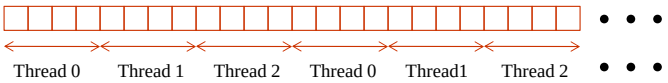
will result in the layout:

Thread 0	Thread 1	Thread 2
X		
Y[0]	Y[1]	Y[2]
z	z	z
A[0][0]	A[0][1]	A[0][2]
A[1][0]	A[1][1]	A[1][2]
A[2][0]	A[2][1]	A[2][2]
A[3][0]	A[3][1]	A[3][2]

6

Blocking of Shared Arrays

- Shared arrays can be distributed to threads in a round robin fashion with arbitrary block sizes.
- Block size and THREADS determine affinity (in which thread's local shared-memory space a shared data item will reside)
- Default block size is 1
- A block size is specified in the declaration as follows:
  - shared [block-size] array[N];
  - e.g.: shared [4] int a[64];



7

Shared Arrays

- Elements of shared arrays are distributed in a round robin fashion, by chunks of block-size, such that the *i*-th element has affinity with thread

  (floor (i/block size) mod THREADS).
- The layout qualifier dictates the blocking factor. This factor is the nonnegative number of consecutive elements
- If the optional constant expression specifying the block size is 0 or not specified (i.e. []), this indicates an indefinite blocking factor where all elements have affinity to the same thread.
- If there is no layout qualifier, the blocking factor (block size) defaults to [1].
- If the layout qualifier is of the form '[*]', the shared object is distributed as if it had a block size of

$$\frac{\frac{sizeof(a)}{upc\_elemtnsizeof(a)} + THREADS - 1}{THREADS}$$

8

Work Sharing with upc_forall()

- Distributes iterations so that each thread gets a bunch of iterations
- Simple C-like syntax and semantics

```
upc_forall( init ; test ; loop ; affinity)
        Statement ;
```

- The affinity field can be an integer expression or a shared reference
- When *affinity* is an integer expression, the loop body of the upc forall statement is executed for each iteration in which the value of MYTHREAD equals the value *affinity* mod THREADS. That is, the upc_forall loop is compiled as:

```
for ( init ; test ; loop)
   if (MYTHREAD == affinity % THREADS)
        Statement ;
```

9

Work Sharing with upc_forall()

- When *affinity* is a reference to a shared item, the loop body of the upc forall statement is executed for each iteration in which the value of MYTHREAD equals the value of upc threadof(*affinity*). Each iteration of the loop body is executed by precisely one thread (the thread that owns the shared variable specified by *affinity*). This is the "**owner computes rule**". In other words, the upc_forall loop is compiled as:

```
for ( init ; test ; loop)
   if (MYTHREAD == the owner thread of (affinity))
        Statement ;
```

- When *affinity* is **continue** or not specified, each loop body of the upc forall statement is performed by every thread.

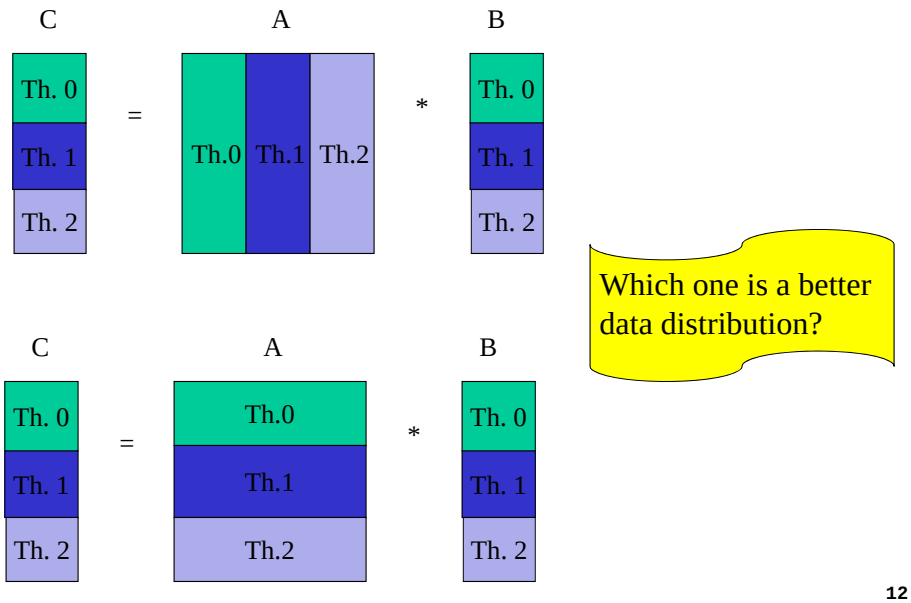
10

Example:UPC Matrix-Vector Multiplication

```
shared int a[THREADS][THREADS] ;
shared int b[THREADS], c[THREADS] ;
void main (void) {
        int i, j;
        upc_forall( i = 0 ; i < THREADS ; i++; i) {
                c[i] = 0;
                for ( j= 0 ; j < THREADS ; j++)
                        c[i] += a[i][j]*b[j];
        }
}
```

11

Data Distribution



12

Example: UPC Matrix-Vector Multiplication (the Better Distribution)

```
shared [THREADS] int a[THREADS][THREADS];
shared int b[THREADS], c[THREADS];

void main (void) {
        int i, j;
        upc_forall( i = 0 ; i < THREADS ; i++; i) {
                c[i] = 0;
                for ( j= 0 ; j< THREADS ; j++)
                        c[i] += a[i][j]*b[j];
        }
}
```

Equivalent to &c[i] or &b[i] since thread i is the owner of c[i] and b[i]

13

UPC Pointers

- Pointer to shared data can be declared as follows:

```
shared int *p;
```

- p is a pointer to an integer residing in the shared memory space (called a pointer to share)
- Example:

```
#define N 100*THREADS
    shared int v1[N], v2[N], v1plusv2[N];
    void main()
    {
        int i;
        shared int *p1, *p2;
        p1=v1; p2=v2;
        upc_forall(i=0; i<N; i++, p1++, p2++; i)
                v1plusv2[i]=*p1+*p2;
    }
```

14

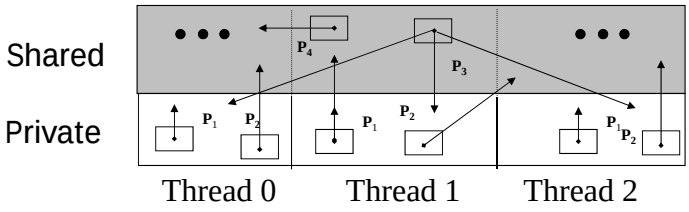
UPC Pointers

Where does it reside?	Where does it point to? Private	Shared
Private	PP	PS
Shared	SP	SS

- How to declare them?
  - int *p1; /* private pointer pointing locally */
  - shared int *p2; /* private pointer pointing to shared space */
  - int *shared p3; /* shared pointer pointing locally */
  - shared int *shared p4; /* shared pointer pointing to shared space */

15

UPC Pointers



- What are the common usages?

int *p1; /* access to private data or to local shared data */

shared int *p2; /* independent access of threads to shared data */

int *shared p3; /* not recommended*/

shared int *shared p4; /* common access of all threads to shared data*/

16

Common Uses for UPC Pointer Types

int *p1;

- These pointers are fast (just like C pointers)
- Use to access local data in part of code performing local work
- Often cast a pointer-to-shared to one of these to get faster access to shared data that is local

shared int *p2;

- Use to refer to remote data
- Larger and slower due to test-for-local + possible communication

int *shared p3;

- Not recommended

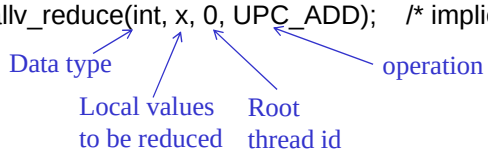
shared int *shared p4;

- Use to build shared linked structures, e.g., a linked list

17

UPC collective functions

- Collective functions are functions that have to be called by every thread and will return the same value to all of them (as a convention, the name of a collective function typically includes “all”)
- Defined by the UPC collectives interface available at
  http://www.gwu.edu/~upc/docs/
- Contains typical functions:
  - Data movement: broadcast, scatter, gather, …
  - Computational: reduce, prefix, …
- Interface has synchronization modes:
  - Barrier before/after is simplest semantics (but may be unnecessary)
- Example:
  - bupc_allv_reduce(int, x, 0, UPC_ADD); /* implies a barriers*/



18

Dynamic allocation of shared memory

- Through a collective function

Number of blocks

Size of block

shared void *upc_all_alloc(size_t nblocks, size_t nbytes);

equivalent to

shared [nbytes] char[nblocks * nbytes].

must be called by all threads and returns the same pointer to all threads.

- Through a global function

shared void *upc_global_alloc(size_t nblocks, size_t nbytes);

Each call to this function allocates a separate memory reagion.

Hence must be called by only one thread.

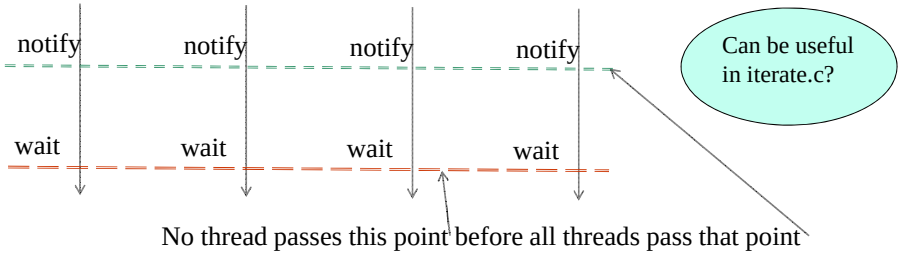
If multiple calls from multiple threads will result in multiple allocations.

19

Syncrhronization

- No implicit synchronization among the threads, but UPC provides mechanisms for Barriers , Locks, Memory Consistency Control and Fences.

- Example: barrier constructs:
  - upc_barrier optional_expr ; /* Blocking Barriers */
  - upc_wait opt_expr; /* wait part of split barrier */
  - upc_notify opt_expr; /* notify part of split barrier */



20