

UPC: Introduction and Recent News

Bill Carlson

IDA Center for Computing Sciences

Kathy Yelick
UC Berkeley

Outline

- ◆ UPC History and Pre-History
- ◆ The Programming Model
 - ◆ Memory Model
 - ◆ Thread/Control Model
 - ◆ Parallel Consistency Model
- ◆ The Language
- ◆ The Libraries -- Including Recent Proposals
- ◆ Current and Future directions
- ◆ Kathy Yelick -- UC Berkley Compiler and Runtime

In The Beginning...

- ◆ C -- The one true light
- ◆ Add Parallelism
 - ◆ Split-C -- Shared Memory from Distributed
 - ◆ AC -- High Performance, Clean Syntax
 - ◆ PCP -- Rich Parallel Control
 - ◆ Many Others
- ◆ Bake Well
 - ◆ UPC!

UPC Design Goals

- ◆ Close to “C Philosophy”
 - ◆ Programming is “performance transparent”
 - ◆ Not too many “safety nets”
- ◆ Fast Implementations
 - ◆ Node Level - use the best compilers
 - ◆ Communication - Great runtimes or better
- ◆ Ubiquitous
 - ◆ Vendors: Cray, HP, Sun
 - ◆ Open Source: IBM, Linux, SGI, Miranet, Quadrics

UPC Highlights

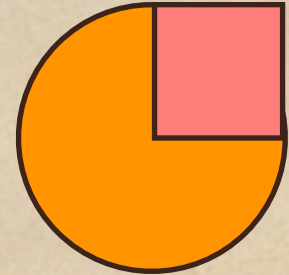
- ◆ Simple Language Extension
 - ◆ Several new keywords
 - ◆ Blocking (HPF-lite)
 - ◆ Clean integration into ANSI/ISO C99
- ◆ Simple Programming Model
 - ◆ Thread Model
 - ◆ One thread per “processor”
 - ◆ Clean synchronization primitives
 - ◆ Memory Model - Private and Shared

Thread Model

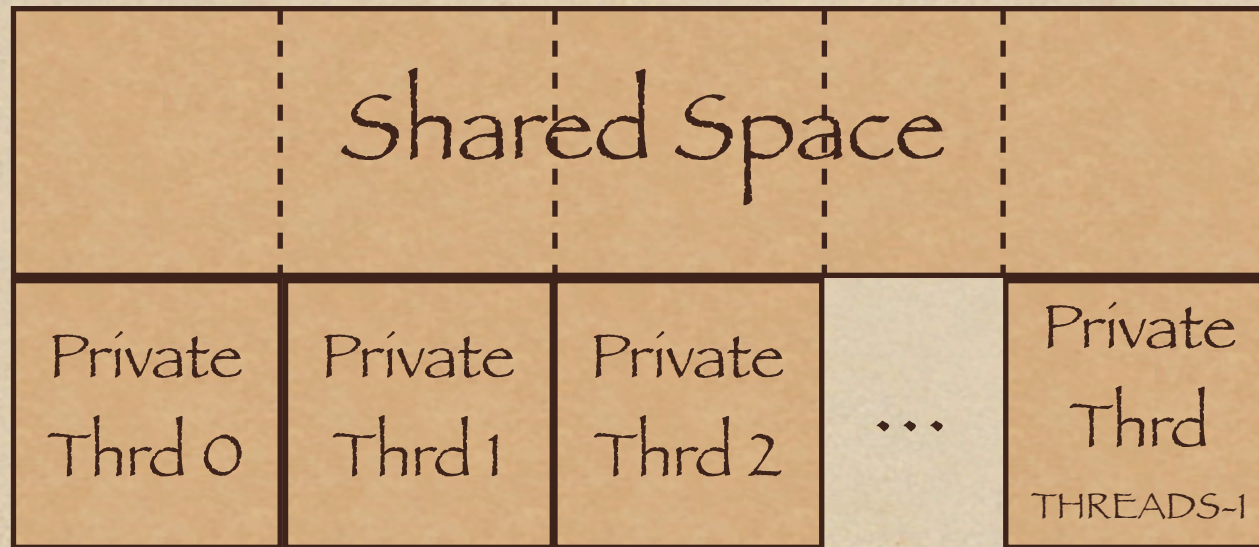
- ◆ THREADS threads, each independent, behaves like a C program
- ◆ MYTHREAD is each thread's index
 - ◆ Range is 0..THREADS-1
- ◆ Barriers
 - ◆ upc_notify; - Announce completion of work
 - ◆ upc_wait; - Wait for others to complete
 - ◆ Work between upc_notify/upc_wait has no guarantees with respect to before/after barrier

Your First UPC Program

```
#include <upc.h>
#include <stdlib.h>
main(int argc, char **argv) {
    int i, hits=0, trials = 1000000; double pi;
    seed48 ((MYTHREAD+1) * THREADS);
    for (i=0; i<trials; i++) hits += hit();
    pi = 4.0*hits/trials;
    printf("Thread %d estimates pi = %g", MYTHREAD, pi);
}
int hit () {
    double x = drand48(), y = drand48();
    return ((x*x+y*y)<1.0); }
```



Memory Model



- ◆ Shared space accessible by any thread
- ◆ Private space accessible by MYTHREAD
- ◆ Static and dynamic allocation, pointers
- ◆ Affinity

Parallel Consistency Model

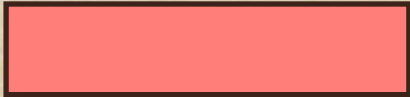
- ◆ Defines an access ordering relationship
 - ◆ Applies only to shared accesses
- ◆ Thread B observes Thread A's accesses
 - ◆ In the order issued if they are "strict"
 - ◆ In any order if they are "relaxed"
- ◆ Ordering applies to accesses, not objects
 - ◆ But specify strict vs. relaxed by declarations

Simple Shared Declarations

- ◆ A new keyword: `shared`
- ◆ Array Declarations, a single object, `shared`
 - ◆ `shared double a[THREADS];`
 - ◆ `shared struct foo b[10][THREADS];`
- ◆ Pointers to shared data items
 - ◆ `shared long *lp;`
 - ◆ Still a private variable, but points to shared
- ◆ Scalar Declarations
 - ◆ `shared int x;`

Your Second UPC Pgm



```
main(int argc, char **argv) {  
    int i, trials = 1000000; double pi;  
    seed48 ((MYTHREAD+1) * THREADS);  
    for (i=0; i<trials; i++) hits += hit();  
      
    pi = 4.0*hits/trials;  
    printf("Thread %d estimates pi = %g", MYTHREAD, pi);  
}
```

Shared Pointer Arithmetic

- ◆ Mainly important for “affinity” concept
 - ◆ Shared references with affinity to MYTHREAD are localized
 - ◆ Array element affinity is dealt like cards, blocking available
- ◆ Shared pointer “+1” points to “next thread”

```
shared int *p, *p1;
```

```
p1 = p + i;
```

- ◆ Becomes

```
p1.thread = (p.thread+i)%THREADS;
```

```
p1.offset = p.offset + ((p.thread+i)/THREADS) * sizeof (*p);
```

Your Third UPC Pgm

```
[REDACTED] /* single writer */  
main(int argc, char **argv) {  
    int i, hits, trials = 1000000; double pi;  
    seed48 ((MYTHREAD+1) * THREADS);  
    for (i=0; i<trials; i++) hits += hit();  
    [REDACTED]  
    upc_barrier;  
    [REDACTED]  
    pi = 4.0*hits/trials;  
    printf("Thread %d estimates pi = %g", MYTHREAD, pi);  
}
```

Shared and Blocked Arrays

- ◆ New syntax on shared declarations

```
shared [block_size] declarator ;
```

- ◆ Element i of a declarator has affinity to thread $(i/\text{block_size}) \bmod \text{THREADS}$

```
shared [3] int x[17];
```

```
shared [3] int *shared [5] y[17];
```

- ◆ Block size may be infinite or automatic

```
shared [] * declarator ;
```

```
shared [*] declarator[100] ;
```

Memory Semantics Control

- ◆ Strict and relaxed apply in declarations

```
strict shared int foo[17];
```

```
relaxed shared int *rp;
```

- ◆ Type casts can override

```
((relaxed shared int *) foo)[12] = 3;
```

- ◆ In “Your Second UPC Pgm”

```
strict int hits;
```

Complex Declarations

```
shared int *ps;          /* pointer to shared */
int *shared sp;         /* shared pointer */
shared int *shared sps; /* shared pointer to shared */
shared in *aps[10];     /* array of 10 pointers to shared */
shared int (*pas)[10];  /* pointer to array of 10 shared */
relaxed shared [12] int *strict shared[17] complex[100];
```

upc_forall Construct

- ◆ Collective “for” loop, all threads get a piece
- ◆ Affinity based work distribution
- ◆ Syntax and semantics similar to C:

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

- ◆ Example:

```
shared int a[100], b[100], c[100]; int i;
upc_forall (i=0; i<100; i++; &a[i])
    a[i] = b[i] + c[i];
```

UPC Standard Library

- ◆ Memory Allocation

- ◆ Global shared space
- ◆ “Local” shared space
- ◆ Collective or distinct allocation

- ◆ Locks

- ◆ Allocated lock in their own space

`upc_lock(); upc_lock_attempt(); upc_free();`

- ◆ Bulk copy

`upc_memget (void *dst, shared const void *src, size_t n);`
`upc_mempup(); upc_memcpy(); upc_memset();`

UPC Collectives Library

- ◆ Relocalization

- ◆ `upc_all_broadcast` (`dst`, `src`, `nbytes`, `sync`)
- ◆ `scatter`, `gather`, `exchange`, `permute`
- ◆ `sync` parameter describes a contract
 - ◆ caller guarantees on entry
 - ◆ callee guarantees on exit

- ◆ Computational

- ◆ `upc_all_reduceT` (`src`, `op`, `nelem`, `bsize`, `func`, `sync`)
- ◆ `upc_all_sort` (`array`, `esize`, `nelem`, `bsize`, `func`, `sync`)

UPC I/O Library

- ◆ UPC base provides thread-parallel I/O
- ◆ UPC-IO provides parallel I/O
 - ◆ In collaboration with MPI-IO at ANL
- ◆ I/O with shared or private buffers
- ◆ I/O with shared or private file pointers
- ◆ List I/O

UPC Library Status

- ◆ Base in all implementations
- ◆ Collective Library
 - ◆ Spec approved
 - ◆ Reference implementation done (first cut)
- ◆ I/O Library
 - ◆ Reference implementation in progress
- ◆ Consensus without unbounded growth

UPC Library Future

- ◆ Synchronization library
 - ◆ Shared work queues
 - ◆ Test-and-set, fetch-and-add
 - ◆ Critical regions
 - ◆ Subset barriers (teams)

Conclusions

- ◆ UPC Implements a shared memory model
 - ◆ Easier to program
 - ◆ Takes advantage of good networks
 - ◆ Runs on all networks
- ◆ UPC is becoming available
 - ◆ Implementations from vendors
 - ◆ Open Source
- ◆ More Info:
 - ◆ <http://upc.gwu.edu> (Overall UPC page)
 - ◆ <http://upc.lbl.gov> (Open Source System)
 - ◆ <http://www.co-array.org> (fortran 90 based)
 - ◆ <http://titanium.cs.berkeley.edu> (java based)