Language Design in UPC

Steven Vormwald
PGAS Seminar 1/22/2009
Contents

• Introduction
• Questionable Content
  – Multidimensional Arrays
  – Generic Functions
  – Shared Pointer Casting
  – Miscellaneous
• Conclusion
Introduction

- What is C?
- Why C?
- What is UPC?
- Why UPC?
UPC Shared Memory
Shared Arrays

- Multidimensional arrays?
  - People complain that Java doesn’t have “real” multidimensional arrays...does UPC?

```c
#define CELL_SIZE DIMZ*DIMY*DIMX
typedef struct gridcell_s {
    double cell[CELL_SIZE];
} gridcell_t;
shared gridcell_t cell_grids[2][THREADS];
#define grids(gridno, z, y, x) \
    cell_grids[gridno][((z)/DIMZ)*NO_ROWS*NO_COLS + \
        ((y)/DIMY)*NO_COLS + \
        ((x)/DIMX)].cell[((z)%DIMZ)*DIMY*DIMX + \
        ((y)%DIMY)*DIMX + \
        ((x)%DIMX)]
```

- We shouldn’t need to abuse the preprocessor to distribute shared arrays properly!

(Code from book “UPC Distributed Shared Memory Programming”)
Shared Pointers

- Generic Object Manipulation?

```c
void fs(shared struct X *x) {
    x->member = CONST;
}
void fl(struct X *x) {
    x->member = CONST;
}
```

- Why do programmers need to care about whether the object is private or shared?
Shared Pointers

- Dynamic Blocksize Casting?

```c
int f(int x, shared void *B) {
  return *((shared [x] int *)B)+x)
}
```

- Would make writing a generic shared-to-shared memcpy-like function a lot easier!
A Small C Function

int A[ROWS][COLS];
int *B = &A[0][0];
int (*C)[COLS] = &A[0];
int i, j;

for(i=0; i<ROWS; i++) {
    C++;
    for(j=i*i+1; j>0; j--) {
        if(++B>=C) {
            printf("B caught C!\n");
            return;
        }
    }
}

}
Shared Memory Vs. C Memory

• C assumes a flat memory layout – comparisons between pointers are defined.
  – All memory addresses have a single distinct integer representation (not necessarily in the range of any integer type...), used when comparing pointers, regardless of their types.

• UPC shared memory is **NOT** flat. Comparisons are defined only on pointers of the same type.
  – Have to be careful when using shared pointers to get different “views” of a shared array...when it is even possible.
Miscellaneous

• What does “\(x=1\)” do (atomically)?

• How should the standard C I/O functions work?

• How can we synchronize groups of threads?

• How do we interact with external (non-UPC) libraries?
Conclusion

• UPC forces programmers to consider low-level implementation details.

• Operations on UPC shared objects are similar to operations on standard C objects, but not identical.

• Many important details are left out of the specification, or are “implementation defined”.