Overview

- Brief overview of
  - CAF
  - UPC
  - HPF (see previous lectures)
- Further reading
Global Address Space (GAS)

- Global address space languages take advantage of
  - Ease of programmability of shared memory parallel
  - SPMD parallelism
  - Allow local-global distinction of data, because data layout matters for performance

- Global address space is logically shared, physically distributed
  - Shared arrays are distributed over processor memories
  - Implicit communication for remote data access

```
Processor1  Processor2                ProcessorN
X(1)        X(2)        .............  X(N)
```
Partitioned Global Address Space (PGAS)

- Global address space with two-level model that supports locality management
  - Local memory (private variables)
  - Remote memory (shared variables)

```c
shared int X[P];
int *ptr = &X[1];
int n = ...
```

```
X[0]  X[1]  ..........  X[P-1]
ptr= n=1  ptr= n=5  ..........  ptr= n=3
```

`Processor1`  `Processor2`  `ProcessorP`
Partitioned Global Address Space (PGAS)

- Global address space with two-level model that supports locality management
  - Local memory (private variables)
  - Remote memory (shared variables)
- Programmer controls critical decisions
  - Data partitioning
  - Communication
- Suitable for mapping to a range of parallel architectures
  - Shared memory, message passing, and hybrid
- Languages: CAF (Fortran), UPC (C), Titanium (Java)
PGAS Model

- Abstract model: partitioned global address space
- Different implementations:
  - Address space partitioned by processors
    - Physically: at the memory address level (= DSM, e.g. Cray T3D/E)
    - Logically: at the variable level, where each variable can be arbitrarily placed in local memory on remote processor
  - Local caching of remote memory?
    - Coherence protocol
  - Communication
    - One-sided, e.g. DMA, is usually faster
    - Two-sided, e.g. MPI send/recv
  - Bulk memory copy operations or individual copies
Co-Array Fortran (CAF)

- Explicitly-parallel extension of Fortran 90/95
  - Commercial compiler from Cray/SGI
  - Open source compiler from Rice University

- Partitioned global address space SPMD with two-level model that supports locality management
  - Local memory (private variables)
  - Remote memory (shared variables)

- Programmer controls critical decisions
  - Data partitioning
  - Communication
A co-array is an array extended with an *image* dimension

```
REAL, DIMENSION(N)[*] :: X,Y
X[::] = Y[Q]
```
CAF: Array Syntax and Implicit Remote Memory Operations

REAL, DIMENSION(N) :: X
REAL, DIMENSION(N) [*] :: Y
REAL, DIMENTION(N,P) [*] :: Z

X = Y[PE] ! get from Y[PE]
Y[PE] = X ! put into Y[PE]
Y[: ] = X ! broadcast X
Y[LIST] = X ! broadcast X over subset
! of PE's in array LIST
Z(: ) = Y[: ] ! all-gather, collect all Y
S = MINVAL(Y[: ] ) ! min (reduce) all Y
Z[: ] = S ! S scalar, promoted to array
! of shape (1:N,1:P)
CAF: Synchronization

COMMON/XCTILB4/ B(N,4)[*]
SAVE /XCTILB4/

ME = THIS_IMAGE()
IF (ME > 1 .AND. ME < NUM_IMAGES()) THEN
   CALL SYNC_ALL( WAIT=(/ME-1,ME+1/) )
   B(:,1) = B(:,3)[ME-1]
   B(:,4) = B(:,2)[ME+1]
   CALL SYNC_ALL( WAIT=(/ME-1,ME+1/) )
ENDIF

Wait for processors on the left and right
Unified Parallel C (UPC)

- UPC is an explicit extension of ANSI C
  - Commercial compilers from Cray/SGI, HP
  - Open source compiler from LBNL/UCB/MTU/UF and GCC-UPC project
- Follows the C language philosophy
  - Programmers are clever and careful and may need to work close to the hardware level
    - to get performance,
    - but can get into trouble!
  - Concise and efficient syntax
- UPC is a PGAS language
  - Global address space with private and shared variables
  - Private/shared pointers to private/shared variables
  - Array data distributions (block/cyclic)
  - Forall worksharing loops
  - Barriers and locks
  - Bulk copy operations between shared and private memory
UPC: Shared Variables

- Private by default
  - C variables and objects are allocated in private memory space for each thread
- Shared variables are explicitly declared and allocated once (by thread 0)
  - Shared variables must be “globally” declared (i.e. static)

```c
shared int ours;
int mine;
```

```
Global address space

ours

mine   mine   ............   mine

Processor1  Processor2  ProcessorP
```
UPC: Simple Example Monte Carlo pi Calculation

```c
int hit()
{
    int const rand_max = 0xFFFFFFFF;
    double x = ((double) rand()) / RAND_MAX;
    double y = ((double) rand()) / RAND_MAX;
    return ((x*x + y*y) <= 1.0);
}
```

Randomly throw darts at (x,y) positions in a unit circle, if $x^2 + y^2 \leq 1$, then point is inside circle

Compute ratio of points inside/total, then $\pi = 4 \times \text{ratio}$
UPC: Simple Example Monte Carlo pi Calculation

```c
#include <upc.h>
shared int hits = 0;
main()
{
    int i;
    int my_trials, trials = ...;
    my_trials = (trials + THREADS - 1)/THREADS;
    srand(MYTHREAD*17);
    for (i=0; i < my_trials; i++)
        hits += hit();
    if (MYTHREAD == 0)
        printf("pi estimated to %g\n", 4*(double)hits/(double)trials);
}
```

**What can go wrong?**

- Divide the work
- Score hits
UPC: Simple Example Monte Carlo pi Calculation

```c
shared int hits = 0;
main()
{ int i, my_trials, trials = ...;
    upc_lock_t *hit_lock = upc_all_lock_alloc();
    my_trials = (trials + THREADS - 1)/THREADS;
    srand(MYTHREAD*17);
    for (i=0; i < my_trials; i++)
    { upc_lock(hit_lock);
        hits += hit();
        upc_unlock(hit_lock);
    }
    upc_barrier;
    if (MYTHREAD == 0)
        printf(“pi estimated to %g\n”,
            4*(double)hits/(double)trials);
    upc_lock_free(hit_lock);
}
```
UPC: Simple Example Monte Carlo pi Calculation

```c
shared int hits[THREADS] = { 0 };  
main()
{ int i, my_trials, trials = ...;  
  my_trials = (trials + THREADS - 1)/THREADS;  
  srand(MYTHREAD*17);  
  for (i=0; i < my_trials; i++)  
    hits[MYTHREAD] += hit();  
  upc_barrier;  
  if (MYTHREAD == 0)  
    for (i=1; i < THREADS; i++)  
      hits[0] += hits[i];  
    tot_trials = THREADS*my_trials;  
    printf("pi estimated to %g\n",  
           4*(double)hits[0]/(double)tot_trials);  
}  
```
UPC: Forall Work Sharing

```
shared int v1[N], v2[N], sum[N];

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

Affinity: here it forces owner-computes rule

Default distribution: cyclic

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

Assume THREADS=4

Elements with affinity to processor 0 are red
UPC: Pointers

Where does the pointer reside?

<table>
<thead>
<tr>
<th></th>
<th>Local</th>
<th>Shared</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private</td>
<td>PP (p1)</td>
<td>PS (p3)</td>
</tr>
<tr>
<td>Shared</td>
<td>SP (p2)</td>
<td>SS (p4)</td>
</tr>
</tbody>
</table>

Where does the referenced value reside?

int *p1;        /* private pointer to local memory */
shared int *p2; /* private pointer to shared space */
int *shared p3; /* shared pointer to local memory */
shared int *shared p4; /* shared pointer to shared space */

Shared pointer to private local memory is not recommended
int *p1;     /* private pointer to local memory */
shared int *p2; /* private pointer to shared space */
int *shared p3; /* shared pointer to local memory */
shared int *shared p4; /* shared pointer to shared space */
UPC: Pointer Example

```c
shared int v1[N], v2[N], sum[N];

int i;
shared int *p1, *p2;

p1 = v1;
p2 = v2;
upc_forall(i=0; i<N; i++, p1++, p2++; i)
    sum[i] = *p1 + *p2;
```


**UPC: Pointers**

- In UPC pointers to shared objects have three fields:
  - thread number
  - local address of block (for blocked data distributions)
  - phase (specifies position in the block)

<table>
<thead>
<tr>
<th>Phase</th>
<th>Thread</th>
<th>Virtual Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>49</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
UPC: Shared Variable Layout

- Non-array share variables have affinity with thread 0
- Array layouts are cyclic or blocked:

```c
shared double x[n];  /* cyclic */  
shared [b] double y[n];  /* blocked */
```

where \( b \) is the block size

- For blocked layouts, element \( i \) has affinity with thread:

\[
(i/b) \mod \text{THREADS}
\]

therefore use \( i/b \) in `upc forall` (owner-computes):

```c
upc forall(i=0; i<N; i++; i/b) y[i] = ...
```
UPC: Consistency Model

- The consistency model of shared memory accesses are controlled by qualifiers
  - Strict: will always appear in order
  - Relaxed: may appear out of order to other threads

- Use strict on variables that are used as synchronization

  ```
  strict: {
    x = y;
    z = y+1;
  }
  ```

- Select the default consistency model with:
  - `#include <upc_strict.h>`
  - `#include <upc_relaxed.h>`
UPC: Fence

- UPC provides a fence construct
  - Syntax
    
    upc_fence;
    equivalent to a null strict reference
    strict { }
  - Ensures that all shared references issued before the
    upc_fence are complete
Further Reading

- CAF: [www.co-array.org](http://www.co-array.org)
- UPC: [upc.gwu.edu](http://upc.gwu.edu)