Acknowledgments

- Supercomputing 2007 tutorial on “Programming using the Partitioned Global Address Space (PGAS) Model” by Tarek El-Ghazawi and Vivek Sarkar
Places in HJ

*here* = place at which current task is executing

place.MAX_PLACES = total number of places (runtime constant)

Specified by value of p in runtime option, `-places p:w`

place.factory.place(i) = place corresponding to index i

`<place-expr>.toString()` returns a string of the form "place(id=0)"

`<place-expr>.id` returns the id of the place as an int

async at(P) S

- Creates new task to execute statement S at place P
- async S is equivalent to async at(here) S
- Main program task starts at place.factory.place(0)

Note that *here* in a child task refers to the place P at which the child task is executing, not the place where the parent task is executing

Example of –places 4:2 option on a SUG@R node (4 places w/ 2 workers per place)

// Main program starts at place 0
async at(place.factory.place(0)) S1;
async at(place.factory.place(0)) S2;
async at(place.factory.place(1)) S3;
async at(place.factory.place(1)) S4;
async at(place.factory.place(1)) S5;
async at(place.factory.place(2)) S6;
async at(place.factory.place(2)) S7;
async at(place.factory.place(2)) S8;
async at(place.factory.place(3)) S9;
async at(place.factory.place(3)) S10;
Example HJ program with places

```java
class T1 {
    final place affinity;
    // T1's constructor sets affinity to place where instance was created
    T1() { affinity = here; ... }
}

finish { // Inter-place parallelism
    System.out.println("Parent..place.=.", here); // Parent task's place
    for (T1 a = . . . ) {
        async at (a.affinity) { // Execute async at place with affinity to a
            a.foo();
            System.out.println("Child..place.=.", here); // Child task's place
        } // async
    } // for
} // finish

public void runDistChunkedForkJoin(int iterations,
    int numChunks, dist d) {
    for (int iter = 0; iter < iterations; iter++) {
        finish for (point [jj] : [0:numChunks-1])
            async at(d.get(jj)) {
                for (point [j] : getChunk([1:n], numChunks, jj))
                    myNew[j] = (myVal[j-1] + myVal[j+1]) / 2.0;
            } // finish-for-async
    } // for iter
} // runDistChunkedForkJoin
```

- Chunk jj is always executed in the same place for each iteration, iter
- Method runDistChunkedForkJoin can be called with values of distribution parameter d
Distributions --- hj.lang.dist

- A distribution maps points in a rectangular index space (region) to places e.g.,
  - \( i \mapsto \text{place.factory.place}(i \mod \text{place.MAX_PLACES}-1) \)

- Programmers are free to create any data structure they choose to store and compute these mappings

- For convenience, the HJ language provides a predefined type, hj.lang.dist, to simplify working with distributions

- Some public members available in an instance \( d \) of hj.lang.dist are:
  - \( d.\text{rank} \) = number of dimensions in the input region for distribution \( d \)
  - \( d.\text{get}(p) \) = place for point \( p \) mapped by distribution \( d \). It is an error to call \( d.\text{get}(p) \) if \( p.\text{rank} \neq d.\text{rank} \).
  - \( d.\text{places}() \) = set of places in the range of distribution \( d \)
  - \( d.\text{restrictToRegion}(pl) \) = region of points mapped to place \( pl \) by distribution \( d \)
Block Distribution

- `dist.factory.block([lo:hi])` creates a block distribution over the one-dimensional region, \( \text{lo:hi} \).
- A block distribution splits the region into contiguous subregions, one per place, while trying to keep the subregions as close to equal in size as possible.
- Block distributions can improve the performance of parallel loops that exhibit spatial locality across contiguous iterations.

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Place id</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

Block Distribution (contd)

- If the input region is multidimensional, then a block distribution is computed over the linearized one-dimensional version of the multidimensional region.
- Example in Table 2: `dist.factory.block([0:7,0:1])` for 4 places

<table>
<thead>
<tr>
<th>Index</th>
<th>0.0</th>
<th>0.1</th>
<th>1.0</th>
<th>1.1</th>
<th>2.0</th>
<th>2.1</th>
<th>3.0</th>
<th>3.1</th>
<th>4.0</th>
<th>4.1</th>
<th>5.0</th>
<th>5.1</th>
<th>6.0</th>
<th>6.1</th>
<th>7.0</th>
<th>7.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Place id</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>
Distributed Parallel Loops

- Listing 2 shows the typical pattern used to iterate over an input region $r$, while creating one async task for each iteration $p$ at the place dictated by distribution $d$ i.e., at place $d.get(p)$.

- This pattern works correctly regardless of the rank and contents of input region $r$ and input distribution $d$ i.e., it is not constrained to block distributions.

```
1 finish {
2   region r = ... // e.g., [0:15] or [0:7,0:1]
3   dist d = dist.factory.block(r);
4   for (point p:r)
5       async at(d.get(p)) {
6           // Execute iteration p at place specified by distribution d
7           ...
8       }
9 } // finish
10 ...
```

Cyclic Distribution

- `dist.factory.cyclic([lo:hi])` creates a cyclic distribution over the one-dimensional region, $lo:hi$.

- A cyclic distribution “cycles” through places $0 \ldots$ place.MAX PLACES – 1 when spanning the input region.

- Cyclic distributions can improve the performance of parallel loops that exhibit load imbalance.

- Example in Table 3: `dist.factory.cyclic([0:15])` for 4 places

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Place id</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

- Example in Table 4: `dist.factory.cyclic([0:7,0:1])` for 4 places

<table>
<thead>
<tr>
<th>Index</th>
<th>0.0</th>
<th>0.1</th>
<th>1.0</th>
<th>1.1</th>
<th>2.0</th>
<th>2.1</th>
<th>3.0</th>
<th>3.1</th>
<th>4.0</th>
<th>4.1</th>
<th>5.0</th>
<th>5.1</th>
<th>6.0</th>
<th>6.1</th>
<th>7.0</th>
<th>7.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Place id</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Figure 1: Cyclic distribution for a 8×8 sized region (e.g., [1:8,1:8]) mapped on to 5 places

Figure source: "Principles of Parallel Programming", Calvin Lin & Lawrence Snyder, http://www.pearsonhighered.com/educator/academic/product/0,3110,0321487907,00.html

Block-Cyclic Distribution

- \texttt{dist.factory.blockCyclic([lo:hi], b)} creates a block-cyclic distribution over the one-dimensional region, \texttt{lo:hi}.

- A block-cyclic distribution combines the locality benefits of the block distribution with the load-balancing benefits of the cyclic distribution by introducing a block size parameter, \texttt{b}.

- The linearized region is first decomposed into contiguous blocks of size \texttt{b}, and then the blocks are distributed in a cyclic manner across the places.

- Example in Table 5: \texttt{dist.factory.blockCyclic([0:15])} for 4 place

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Place id</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Data Distributions

- In HJ, distributions are used to guide computation mappings for affinity.
- The idea of distributions was originally motivated by mapping data (array elements) to processors.
- E.g., Unified Parallel C (UPC) language for distributed-memory parallel machines (Thread = Place).
- A pointer-to-shared can reference all locations in the shared space, but there is data-thread affinity.

Affinities for Shared and Private Data in UPC

Examples of Shared and Private Data Layout:

Assume THREADS = 3

```c
shared int x;  /* x will have affinity to thread 0 */
shared int y[2*THREADS];  /* cyclic distribution by default */
int z;  /* private by default */
```

Will result in the layout:

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y[0]</td>
<td>y[1]</td>
<td>y[2]</td>
</tr>
<tr>
<td>y[3]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>z</td>
<td>z</td>
</tr>
</tbody>
</table>
Shared and Private Data

shared int A[4][THREADS];

will result in the following data layout:

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[0][0]</td>
<td>A[0][1]</td>
<td>A[0][2]</td>
</tr>
</tbody>
</table>

Block-Cyclic Distributions for Shared Arrays

• Default block size is 1
• Shared arrays can be distributed on a block per thread basis, round robin with arbitrary block sizes.
• A block size is specified in the declaration as follows:
  — shared [block-size] type array[N];
  — e.g.: shared [4] int a[16];
Shared and Private Data

Assume THREADS = 4


will result in the following data layout:

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
<th>Thread 2</th>
<th>Thread 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A[3][0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A[3][1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A[3][2]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Announcements (REMINDEER)

- Homework 3 due on Wednesday, Feb 22nd
  - Performance results for parts 2 and 3 of assignment must be obtained on Sugar (see Section 4)

- No lab this week
  - Use the time for HW3 and to prepare for Exam 1

- Exam 1 will be held in the lecture on Friday, Feb 24th
  - Closed book 50-minute exam
  - Scope of exam includes lectures up to Monday, Feb 20th
  - Feb 22nd lecture will be a midterm review before exam