Lecture 33: Task Affinity with Places

Mack Joyner
mjoyner@rice.edu

http://comp322.rice.edu
Cores communicate by reading and writing data in a “shared memory”

- Memory hierarchy for a single Intel Xeon (Nehalem) Quad-core processor chip
  - A NOTS node contains TWO 8-core or 12-core E5-2650 v2 Ivy Bridge chips, for a total of 16 or 24 cores
What is the cost of a Memory Access? 
An example Memory Hierarchy

- **Registers**

- **L0:** Registers

- **L1:** L1 cache (Static RAM)
  - CPU registers hold words retrieved from L1 cache

- **L2:** L2 cache (Static RAM)
  - L1 cache holds cache lines retrieved from L2 cache

- **Main memory (Dynamic RAM)**

- **L3:** Main memory holds disk blocks retrieved from local disks

- **L4:** Local secondary storage (local disks)
  - Local disks hold files retrieved from disks on remote network servers

- **L5:** Remote secondary storage (tapes, distributed file systems, Web servers)

Cache Memories

- **Cache memories** are small, fast SRAM-based memories managed automatically in hardware.
  - Hold frequently accessed blocks of main memory
- CPU looks first for data in caches (e.g., L1, L2, and L3), then in main memory.
- Typical system structure:

Locality

• Principle of Locality:
  — Empirical observation: Programs tend to use data and instructions with addresses near or equal to those they have used recently

• Temporal locality:
  — Recently referenced items are likely to be referenced again in the near future

• Spatial locality:
  — Items with nearby addresses tend to be referenced close together in time
  — A Java programmer can only influence spatial locality at the intra-object level
    - The garbage collector and memory management system determines inter-object placement

Source: http://www.cs.cmu.edu/afs/cs/academic/class/15213-f10/www/lectures/09-memory-hierarchy.pptx
Locality Example

- **Data references**
  - Reference array elements in succession (stride-1 reference pattern).
  - Reference variable sum each iteration.
- **Instruction references**
  - Cycle through loop repeatedly.
  - Reference instructions in sequence.

```java
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

• Memory hierarchy for a single Intel Xeon (Nehalem) Quad-core processor chip
Programmer Control of Task Assignment to Processors

- The parallel programming constructs that we’ve studied thus far result in tasks that are assigned to processors *dynamically* by the HJ runtime system
  - Programmer does not worry about task assignment details
- Sometimes, programmer control of task assignment can lead to significant performance advantages due to improved locality
- Motivation for HJ “places”
  - Provide the programmer a mechanism to restrict task execution to a subset of processors for improved locality
**Places in HJlib**

**HJ programmer defines mapping from HJ tasks to set of places**

HJ runtime defines mapping from places to one or more worker Java threads per place

The API calls

```java
HjSystemProperty.numPlaces.set(p);
HjSystemProperty.numWorkers.set(w);
```

when executing an HJ program can be used to specify

- `p`, the number of places
- `w`, the number of worker threads per place

we will abbreviate this as `p:w`
Example of 4:2 option on an 8-core node
(4 places w/ 2 workers per place)
Places in HJlib

here() = place at which current task is executing
numPlaces() = total number of places (runtime constant)
    Specified by value of p in runtime option:
    HjSystemProperty.numPlaces.set(p);
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
asyncAt(P, () -> S)
    • Creates new task to execute statement S at place P
    • async(() -> S) is equivalent to asyncAt(here(), () -> S)
    • Main program task starts at 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 4:2 option on an 8-core node
(4 places w/ 2 workers per place)

// Main program starts at place 0
asyncAt(place(0), () -> S1);
asyncAt(place(0), () -> S2);

draw_diagram

draw_diagram

asyncAt(place(1), () -> S3);
asyncAt(place(1), () -> S4);
asyncAt(place(1), () -> S5);

asyncAt(place(2), () -> S6);
asyncAt(place(2), () -> S7);
asyncAt(place(2), () -> S8);

asyncAt(place(3), () -> S9);
asyncAt(place(3), () -> S10);
Example of 1:8 option (1 place w/ 8 workers per place)

All async’s run at place 0 when there’s only one place!
HJ program with places

1. private static class T1 {
2.     final HjPlace affinity;
3. 
4.     public T1(HjPlace affinity) {
5.         // set affinity of instance to place where it is created
6.         this.affinity = here();
7.         ...
8.     }
9.     public void foo() { ... }
10. }
11. 
12.     finish(() -> {
13.         println("Parent place: " + here());
14.         for (T1 a : t1Objects) {
15.             // Execute async at place with affinity to a
16.             asyncAt(a.affinity, () -> {
17.                 println("Child place: " + here()); // Child task's place
18.                 a.foo();
19.             });
20.         }
21.     });
Block Distribution

- A block distribution splits the index 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.
- Example: dist.get(index) for a block distribution on 4 places, when index is in the range, 0…15

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Distributed Parallel Loops

- The pseudocode below 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 \( d \)\( \text{get}(p) \).

- 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

```python
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:   ...  
```
**Chunked Fork-Join Iterative Averaging Example with Places**

1. public void runDistChunkedForkJoin(
2.   int iterations, int numChunks, Dist dist) {
3.     // dist is a user-defined map from int to HjPlace
4.     for (int iter = 0; iter < iterations; iter++) {
5.         finish(() -> {
6.             for (0, numChunks - 1, (jj) -> {
7.                 asyncAt(dist.get(jj), () -> {
8.                     for (getChunk(1, n, numChunks, jj), (j) -> {
9.                         myNew[j] = (myVal[j-1] + myVal[j+1]) / 2.0;
10.                     }
11.                  });
12.                 });
13.             });
14.             double[] temp = myNew; myNew = myVal; myVal = temp;
15.         } // for iter
16.     } // for iter

• Chunk jj is always executed in the same place for each iter
• Method runDistChunkedForkJoin can be called with different values of distribution parameter d
Locality benefits will be realized if all instances of chunk 0 execute on the same core and reuse data from the same cache.
Cyclic Distribution

- A cyclic distribution “cycles” through places 0 … place.MAX PLACES – 1 when spanning the input region.
- Cyclic distributions can improve the performance of parallel loops that exhibit load imbalance.
- Example: dist.get(index) for a cyclic distribution on 4 places, when index is in the range, 0…15.

<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>