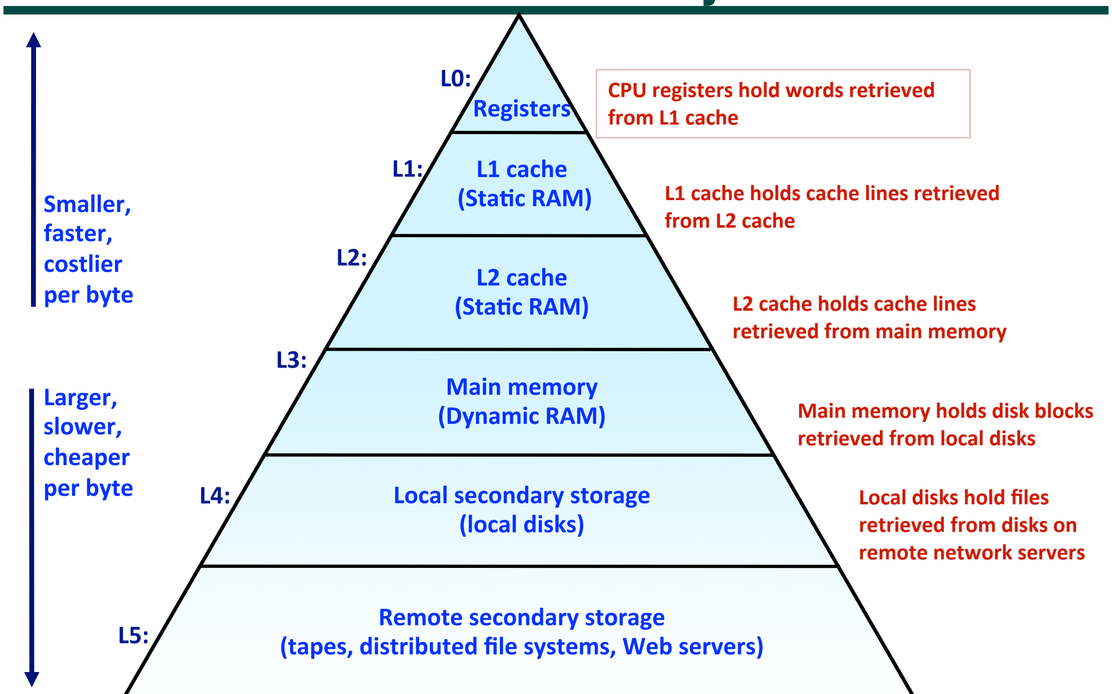
COMP 322: Fundamentals of Parallel Programming

Lecture 17: Task Affinity with Places

Vivek Sarkar
Department of Computer Science, Rice University
vsarkar@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP322

An example Memory Hierarchy --- what is the cost of a Memory Access?



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

Storage Trends

SRAM

Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
$/MB	19,200	2,900	320	256	100	75	60	320
access (ns)	300	150	35	15	3	2	1.5	200

DRAM

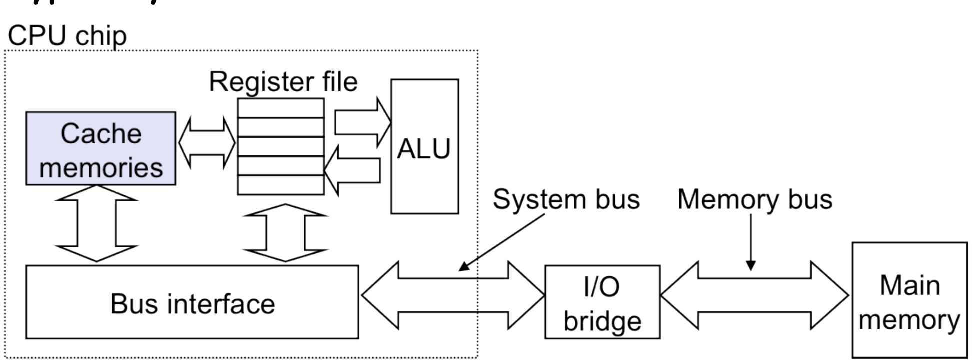
Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
$/MB	8,000	880	100	30	1	0.1	0.06	130,000
access (ns)	375	200	100	70	60	50	40	9
typical size (MB)	0.064	0.256	4	16	64	2,000	8,000	125,000

Disk

Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
$/MB	500	100	8	0.30	0.01	0.005	0.0003	1,600,000
access (ms)		87	75	28	10	8	4	
3	29							
typical size (MB)	1	10	160	1,000	20,000	160,000	1,500,000	1,500,000

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:



Examples of Caching in the Hierarchy

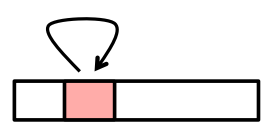
Hierarchy Level	What is cached?	Where is it cached?	Latency (cycles)	Managed by
Registers	4-32 bytes (words)	CPU core	0	Compiler
TLB	Address translations	On-chip TLB	0	Hardware
L1 cache	64-bytes block	On-Chip L1	$O(10^0)$	Hardware
L2 cache	64-bytes block	On/Off-Chip L2	$O(10^1)$	Hardware
Virtual Memory	4 KB page	Main memory	$O(10^2)$	Hardware & OS
Buffer cache	Parts of files	Main memory	$O(10^2)$	OS
Disk cache	Disk sectors	Disk controller	$O(10^5)$	Disk firmware
Network buffer cache	Parts of files	Local disk	$O(10^7)$	AFS/NFS client
Browser cache	Web pages	Local disk	$O(10^7)$	Web browser
Web cache	Web pages	Remote server disks	$O(10^9)$	Web proxy server

Ultimate goal: create a large pool of storage with average cost per byte that approaches that of the cheap storage near the bottom of the hierarchy, and average latency that approaches that of fast storage near the top of the hierarchy.

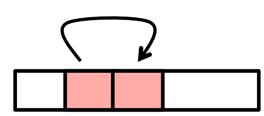
Source: http://www.cs.cmu.edu/afs/cs/academic/class/15213-f10/www/lectures/09-memory-hierarchy.pptx

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

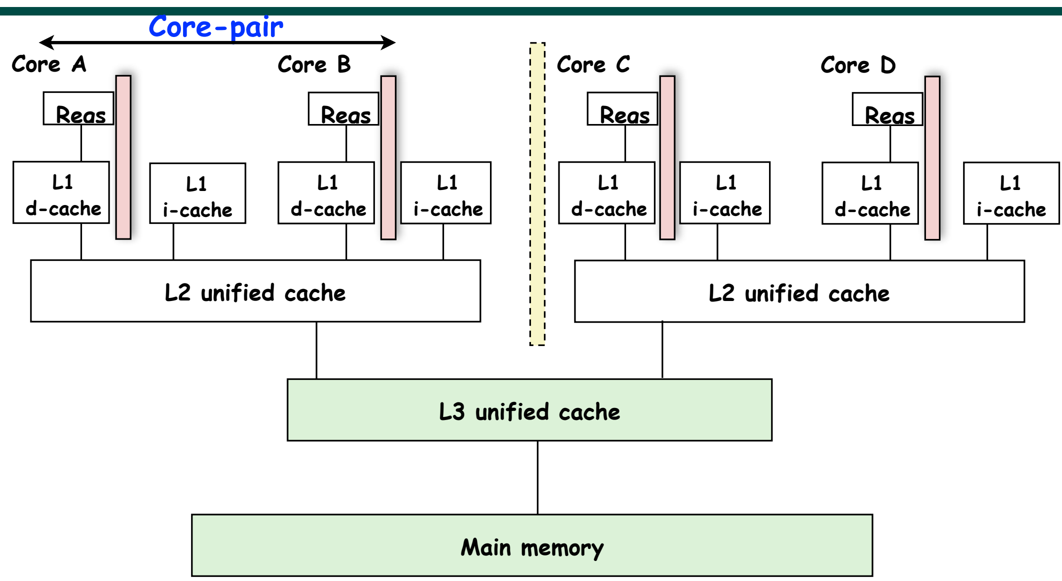
Locality Example

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

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

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

Memory Hierarchy in a Multicore Processor



- Memory hierarchy for a single Intel Xeon Quad-core E5440 HarperTown processor chip
  - — A SUG@R node contains TWO such chips, for a total of 8 cores

Programmer Control of Task Assignment to Processors

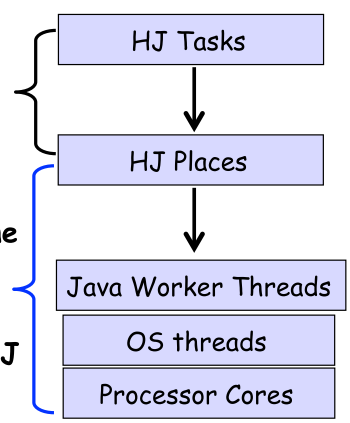
- The parallel programming constructs that we've studied thus far result in tasks that are assigned to processors dynamically by the HJ runtime system
  - —Programmer does not worry about task assignment details
- Sometimes, programmer control of task assignment can lead to significant performance advantages due to improved locality
- Motivation for HJ "places"
  - —Provide the programmer a mechanism to map each task to a set of processors when the task is created

Places in HJ

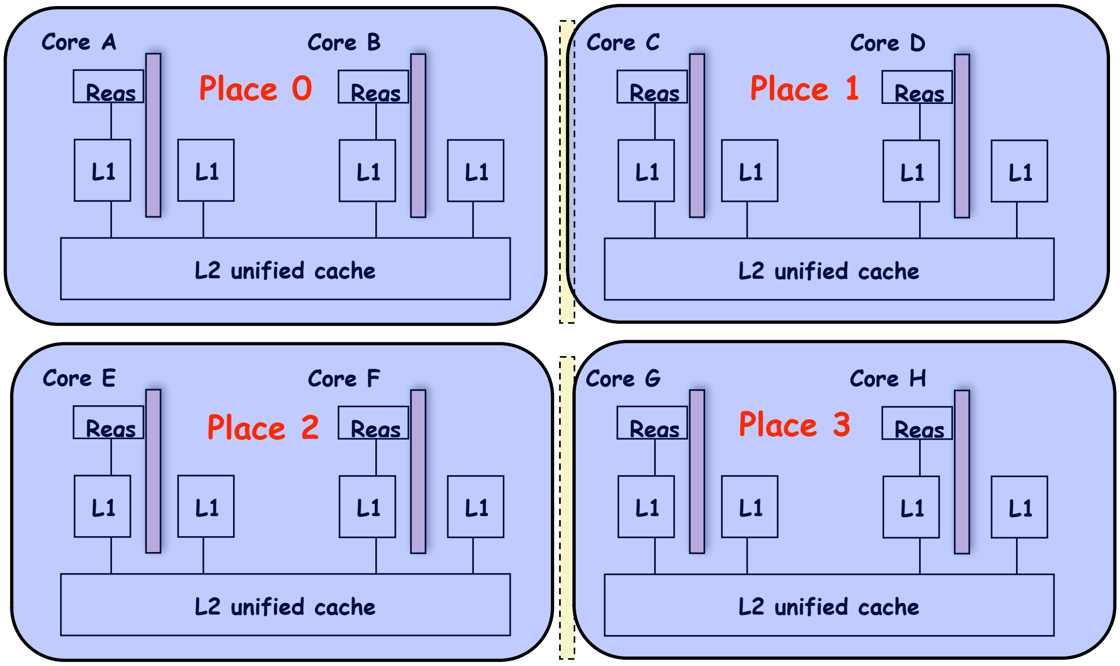
HJ programmer defines mapping from HJ tasks to set of places

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

The option "-places p:w" when executing an HJ program can be used to specify
p, the number of places
w, the number of worker threads per place



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



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

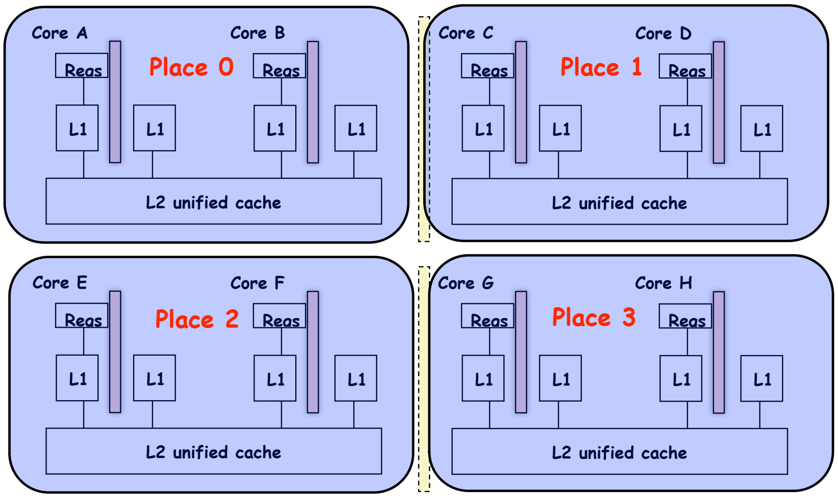
- Creates new task to execute statement S at place P
- **async S** is equivalent to **async at(here) S**
- Main program task starts at **place.factory.place(0)**

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;
```

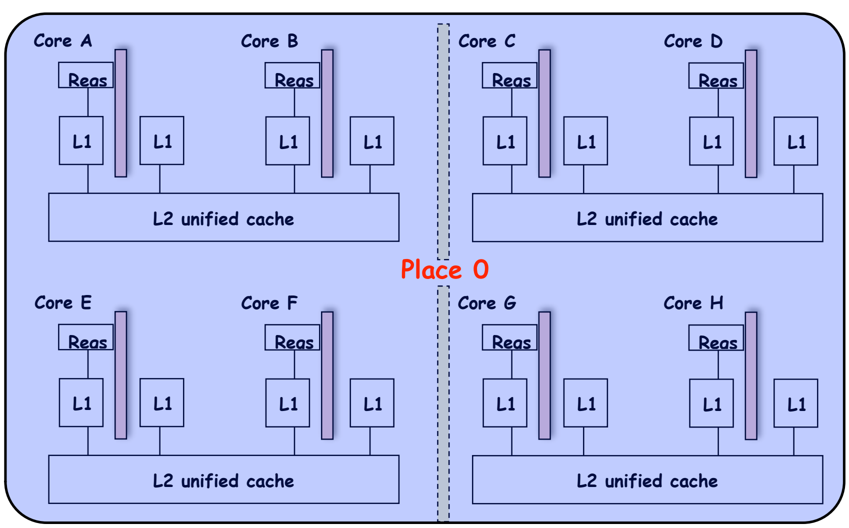



```
async at(place.factory.place(2)) S6;
async at(place.factory.place(2)) S7;
async at(place.factory.place(2)) S8;
```

```
async at(place.factory.place(3)) S9;
async at(place.factory.place(3)) S10;
```


Example of –places 1:8 option (1 place w/ 8 workers per place)

All async's run at place 0 when there's only one place!



Example HJ program with places

```
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
. . .
```

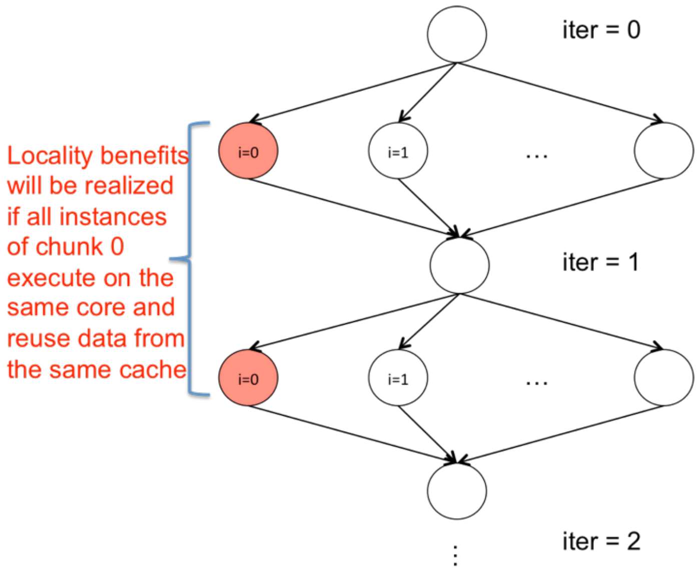

Chunked Fork-Join Iterative Averaging Example with Places

```
for (point [iter] : [0:iterations-1]) {
  finish for (point [i] : [0:tasks-1]) {
    async at(place.factory.place(i % place.MAX_PLACES)) {
      int start = i * batchSize + 1;
      for (point [j] : [start:Math.min(start+batchSize-1,n)]) {
        myNew[j] = (myVal[j-1] + myVal[j+1]) / 2.0;
      }
    } // async
  } // finish for
  double[] temp = myNew; myNew = myVal; myVal = temp;
}
```

- Assume a -places 4:4 configuration with 4 places and 4 workers per places for execution on a 16-core machine
  - Set tasks = 16 so as to create one async per worker
  - Use i % place.MAX_PLACES to compute destination place for each async
  - ➔ Each subarray is processed at same place for successive iterations of for-iter loop

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Place id	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

Analyzing Locality of Fork-Join Iterative Averaging Example with Places



Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Place id	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

Distributions

- A distribution maps points in a rectangular index space (region) to places e.g.,
  - i → place.factory.place(i % 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 as follows
  - d.rank = number of dimensions in the input region for distribution d
  - d.get(p) = place for point p mapped by distribution d. It is an error to call d.get(p) if p.rank != d.rank.
  - d.places() = set of places in the range of distribution d
  - d.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, 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.

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Place id	0				1				2				3			

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

Index	[0,0]	[0,1]	[1,0]	[1,1]	[2,0]	[2,1]	[3,0]	[3,1]	[4,0]	[4,1]	[5,0]	[5,1]	[6,0]	[6,1]	[7,0]	[7,1]
Place id	0				1				2				3			

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

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


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 … 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

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Place id	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

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

Index	[0,0]	[0,1]	[1,0]	[1,1]	[2,0]	[2,1]	[3,0]	[3,1]	[4,0]	[4,1]	[5,0]	[5,1]	[6,0]	[6,1]	[7,0]	[7,1]
Place id	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

Announcements (REMINDER)

- 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 next 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
  - Contact me ASAP if you have an extenuating circumstance and need to take the midterm at an alternate time