CSE502: Foundations of Parallel Programming

Lecture 13: Task Affinity with Hierarchical Place Trees

Vivek Kumar

Computer Science and Engineering

IIIT Delhi

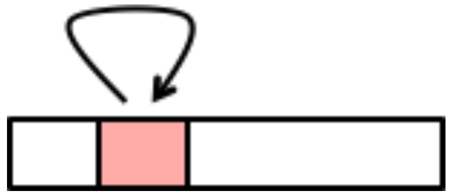
vivekk@iiitd.ac.in

Today's Class

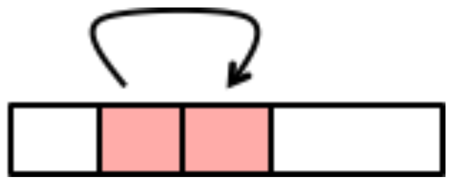
- Task affinity with Hierarchical Place Trees (HPT)

Locality

- Principal of Locality
  - Empirical observation: Processors tend to access same set or nearby memory locations repetitively over a short period of time



- 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

Source: https://wiki.rice.edu/confluence/download/attachments/24426821/comp322-s16-lec32-slides-v1.pdf?version=1&modificationDate=1483206145705&api=v2

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: https://wiki.rice.edu/confluence/download/attachments/24426821/comp322-s16-lec32-slides-v1.pdf?version=1&modificationDate=1483206145705&api=v2

Iterative Averaging with Places – Sequential Version

```
double A[SIZE+2], A_shadow[SIZE+2];

void runSequential() {
  for (uint64_t iter=0; iter<ITERATIONS; iter++) {
    for (uint64_t j=1; j<=SIZE; j++) {
      A_shadow[j] = (A[j-1] + A[j+1])/2.0;
    }
    double* temp = A_shadow;
    A_shadow = A;
    A = temp;
  }
}
```

https://classes.engineering.wustl.edu/cse231/core/index.php/Iterative_Averaging
Code available on github: https://github.com/vivkumar/cse502/blob/master/hclib/test/lec10/

Iterative Averaging with Places – async-finish Version

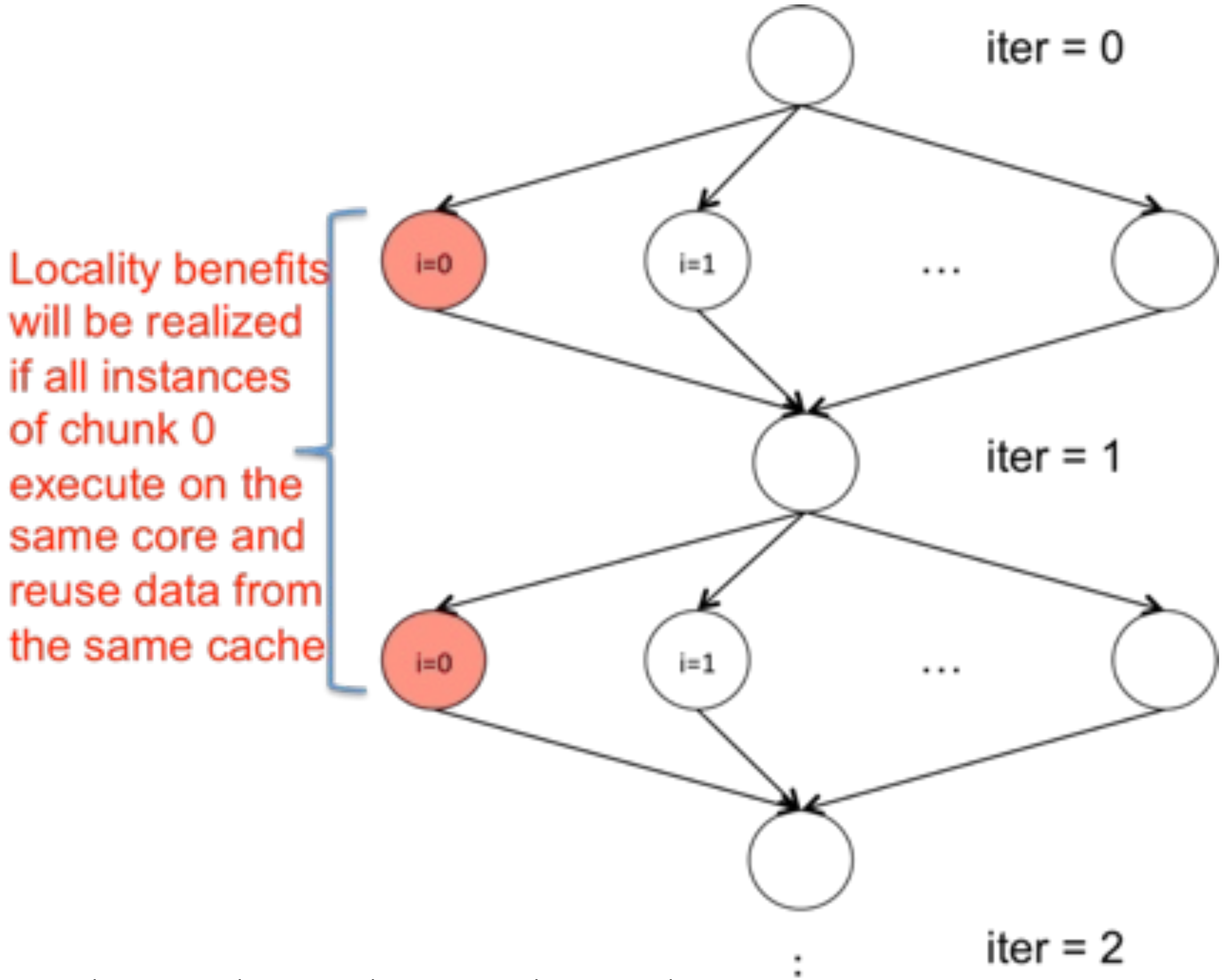
```
double A[SIZE+2], A_shadow[SIZE+2];

void runAsyncFinish() {
  int chunkSize = SIZE / num_workers();
  for (uint64_t iter=0; iter<ITERATIONS; iter++) {
    finish([=]() {
      for (uint64_t i=0; i<num_workers(); i++) {
        async([=]() {
          int start = i * chunkSize + 1;
          int end = start + chunkSize - 1;
          for (uint64_t j=start; j<=end; j++) {
            A_shadow[j] = (A[j-1] + A[j+1])/2.0;
          }
        });
      }
    });
    double* temp = A_shadow;
    A_shadow = A;
    A = temp;
  }
}
```

Does it provide better locality?

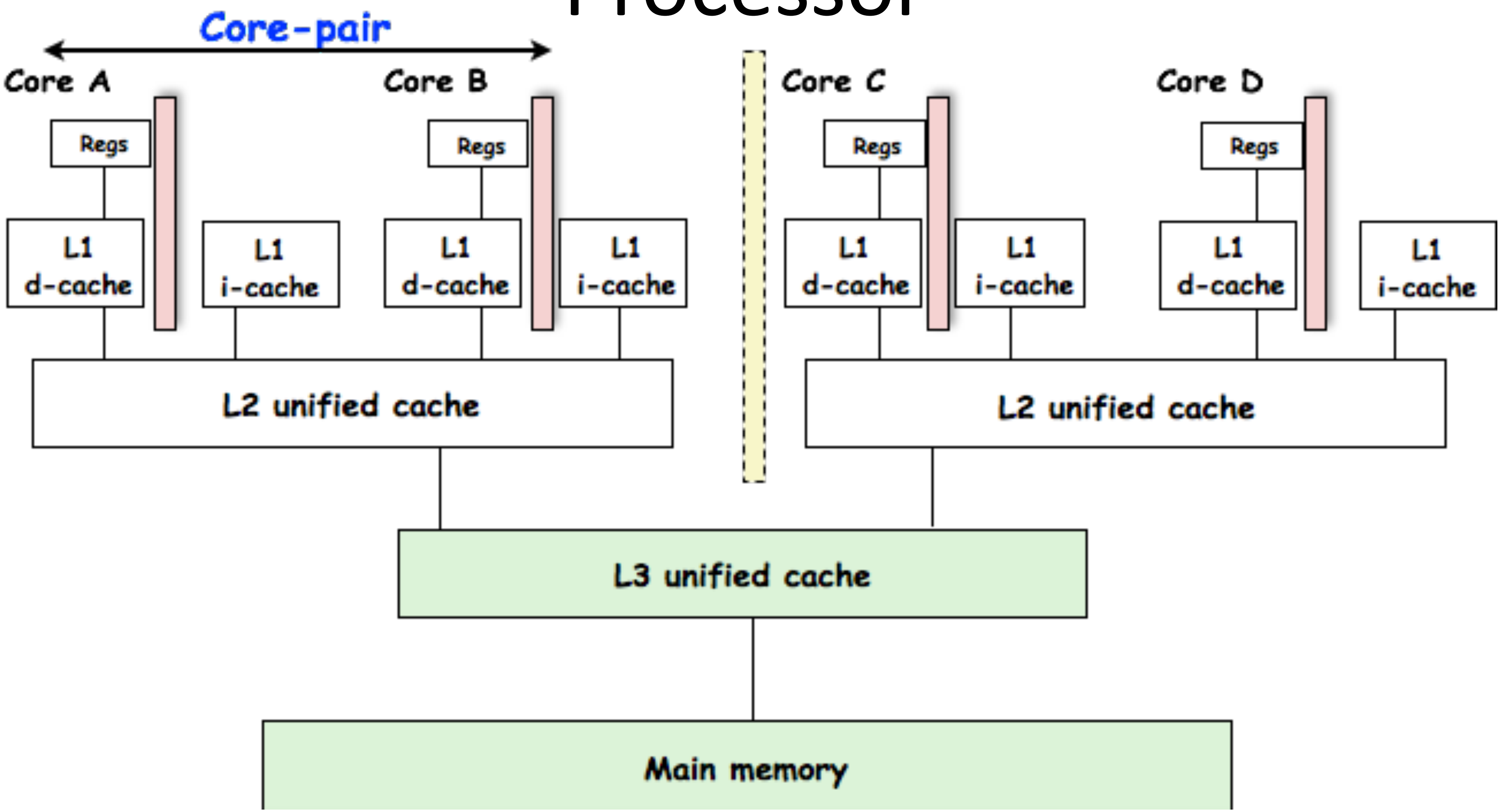
Code available on github: https://github.com/vivkumar/cse502/blob/master/hclib/test/lec10/

Analyzing Locality Iterative Averaging



Source: https://wiki.rice.edu/confluence/download/attachments/24426821/comp322-s16-lec32-slides-v1.pdf?version=1&modificationDate=1483206145705&api=v2

Memory Hierarchy in a Multicore Processor



Memory hierarchy for a single Intel Xeon (Nehalem) Quad-core processor chip

Source: https://wiki.rice.edu/confluence/download/attachments/24426821/comp322-s16-lec32-slides-v1.pdf?version=1&modificationDate=1483206145705&api=v2

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 HClib 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 HClib "places"
  - Provide the programmer a mechanism to restrict task execution to a subset of processors for improved locality

Source: https://wiki.rice.edu/confluence/download/attachments/24426821/comp322-s16-lec32-slides-v1.pdf?version=1&modificationDate=1483206145705&api=v2

Task Affinity

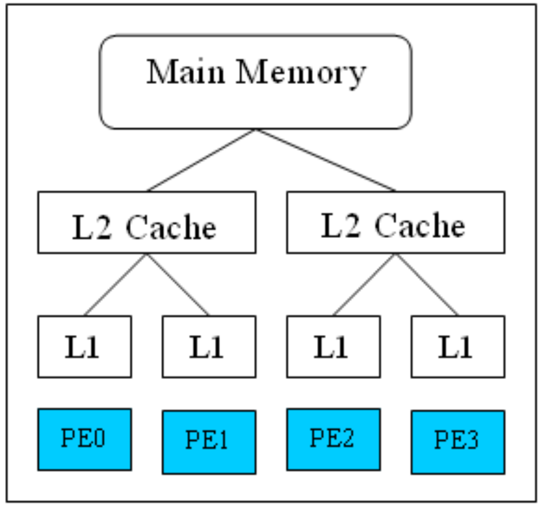
- This is a programming feature provided to the programmer by which he can control the placement of the async tasks in different levels of memory hierarchy
  - Notion of “place” introduced by X10 language
  - Shared memory
    - Habanero-C and Habanero-Java
    - OpenMP does not support this yet but it will be coming up in the near future
  - Distributed memory
    - X10, Chapel, UPC++, HabaneroUPC++

Hierarchical Place Trees in HClib

- Abstraction of the memory hierarchy that a HClib program is executed on (using XML document)
- Place denoting affinity group at memory hierarchy level
  - E.g., L1 cache, L2 cache, DRAM
- Leaf places include worker threads
  - E.g., W0, W1, W2, W3
- Workers can push task to any place

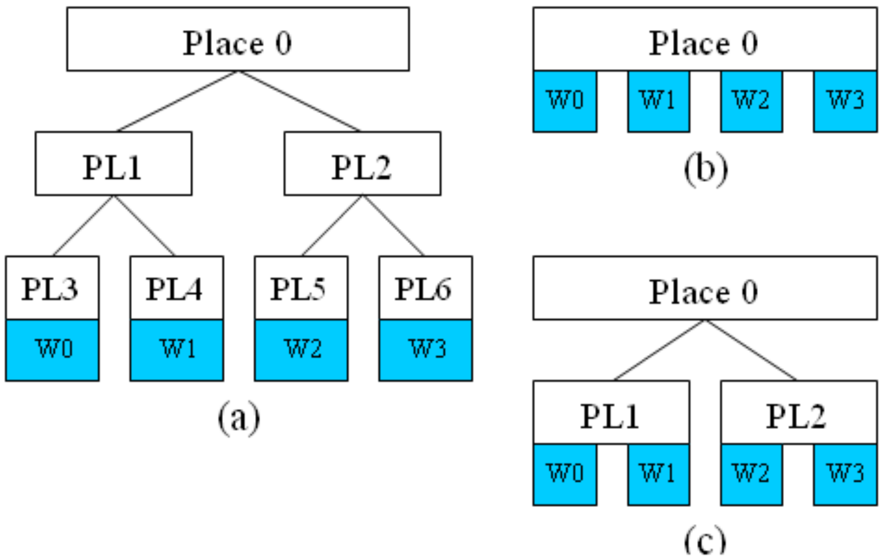
asyncAtHpt(place*, lambda_function)

Example: HPT for a Quad Core Processor



A Quad-core workstation

Three different HPTs possible on this quad core processor



Places in HClib

Some basic APIs in HClib for HPTs

place_t* get_current_place() //place at which current task is executing

int get_num_places(place_type_t type) //total number of places
// (runtime constant)

- type = CACHE_PLACE or MEM_PLACE (accelerator places coming up)

place_t* get_places(places_array, place_type_t type) //array of all
// places of "type"

asyncAtHpt(place_t*, S) //Creates new task to
//execute statement S at place P

A Sample HPT File

```
<?xml version="1.0"?>
<!DOCTYPE HPT SYSTEM "hpt.dtd">

<HPT version="0.1" info="HPT test">
  <place num="1" type="mem">
    <place num="2" type="cache">
      <worker num="1"/>
    </place>
  </place>
</HPT>
```

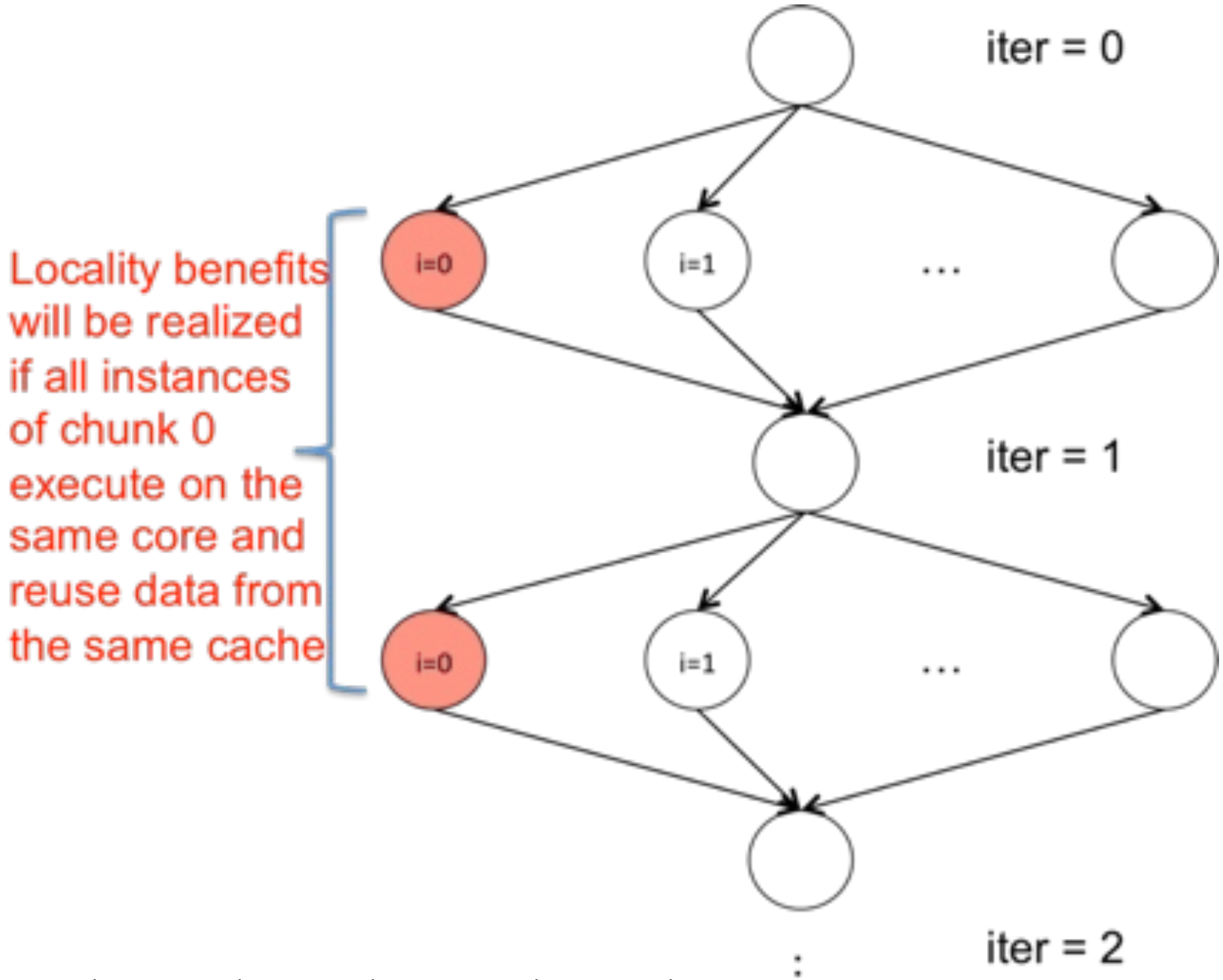
Iterative Averaging with Places – HPT Version

```
double A[SIZE+2], A_shadow[SIZE+2];

void runOnHPT() {
  int numPlaces = get_num_places(place_type_t::CACHE_PLACE);
  place_t** cachePlaces = malloc(sizeof(place_t*) * numPlaces);
  get_places(cachePlaces, place_type_t::CACHE_PLACE);
  int chunkSize = SIZE / numPlaces;
  for (uint64_t iter=0; iter<ITERATIONS; iter++) {
    finish([=]() {
      for (uint64_t i=0; i<num_workers(); i++) {
        asyncAtHpt(cachePlace[i], [=]() {
          int start = i * chunkSize + 1;
          int end = start + chunkSize - 1;
          for (uint64_t j=start; j<=end; j++) {
            A_shadow[j] = (A[j-1] + A[j+1])/2.0;
          }
        });
      }
    });
    double* temp = A_shadow;
    A_shadow = A;
    A = temp;
  }
  free(cachePlaces);
}
```


Code available on github: https://github.com/vivkumar/cse502/blob/master/hclib/test/lec10/

Analyzing Locality of Fork-Join Iterative Averaging Example with Places



Source: https://wiki.rice.edu/confluence/download/attachments/24426821/comp322-s16-lec32-slides-v1.pdf?version=1&modificationDate=1483206145705&api=v2

Performance Analysis of 1D Iterative Averaging with/without HPT

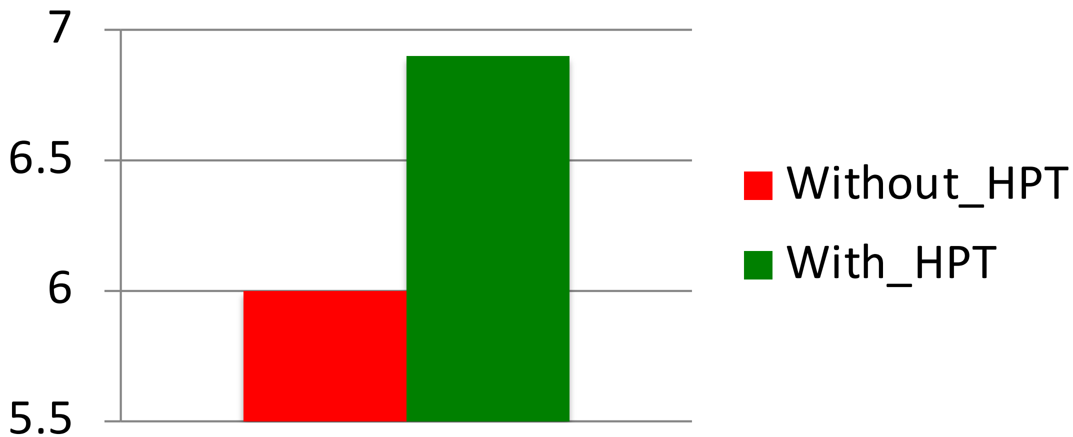
```
double A[SIZE+2], A_shadow[SIZE+2];

void runAsyncFinish() {
  int chunkSize = SIZE / num_workers();
  for (uint64_t iter=0; iter<ITERATIONS; iter++) {
    finish([=]() {
      for (uint64_t i=0; i<num_workers(); i++) {
        async([=]() {
          int start = i * chunkSize + 1;
          int end = start + chunkSize - 1;
          for (uint64_t j=start; j<=end; j++) {
            A_shadow[j] = (A[j-1] + A[j+1])/2.0;
          }
        });
      }
    });
    double* temp = A_shadow;
    temp = A;
    A = temp;
  }
}
```

```
double A[SIZE+2], A_shadow[SIZE+2];

void runOnHPT() {
  int numPlaces = get_num_places(place_type_t::CACHE_PLACE);
  place_t** cachePlaces = malloc(sizeof(place_t*) * numPlaces);
  get_places(cachePlaces, place_type_t::CACHE_PLACE);
  int chunkSize = SIZE / numPlaces;
  for (uint64_t iter=0; iter<ITERATIONS; iter++) {
    finish([=]() {
      for (uint64_t i=0; i<num_workers(); i++) {
        asyncAtHpt(cachePlace[i], [=]() {
          int start = i * chunkSize + 1;
          int end = start + chunkSize - 1;
          for (uint64_t j=start; j<=end; j++) {
            A_shadow[j] = (A[j-1] + A[j+1])/2.0;
          }
        });
      }
    });
    double* temp = A_shadow;
    temp = A;
    A = temp;
  }
  free(cachePlaces);
}
```

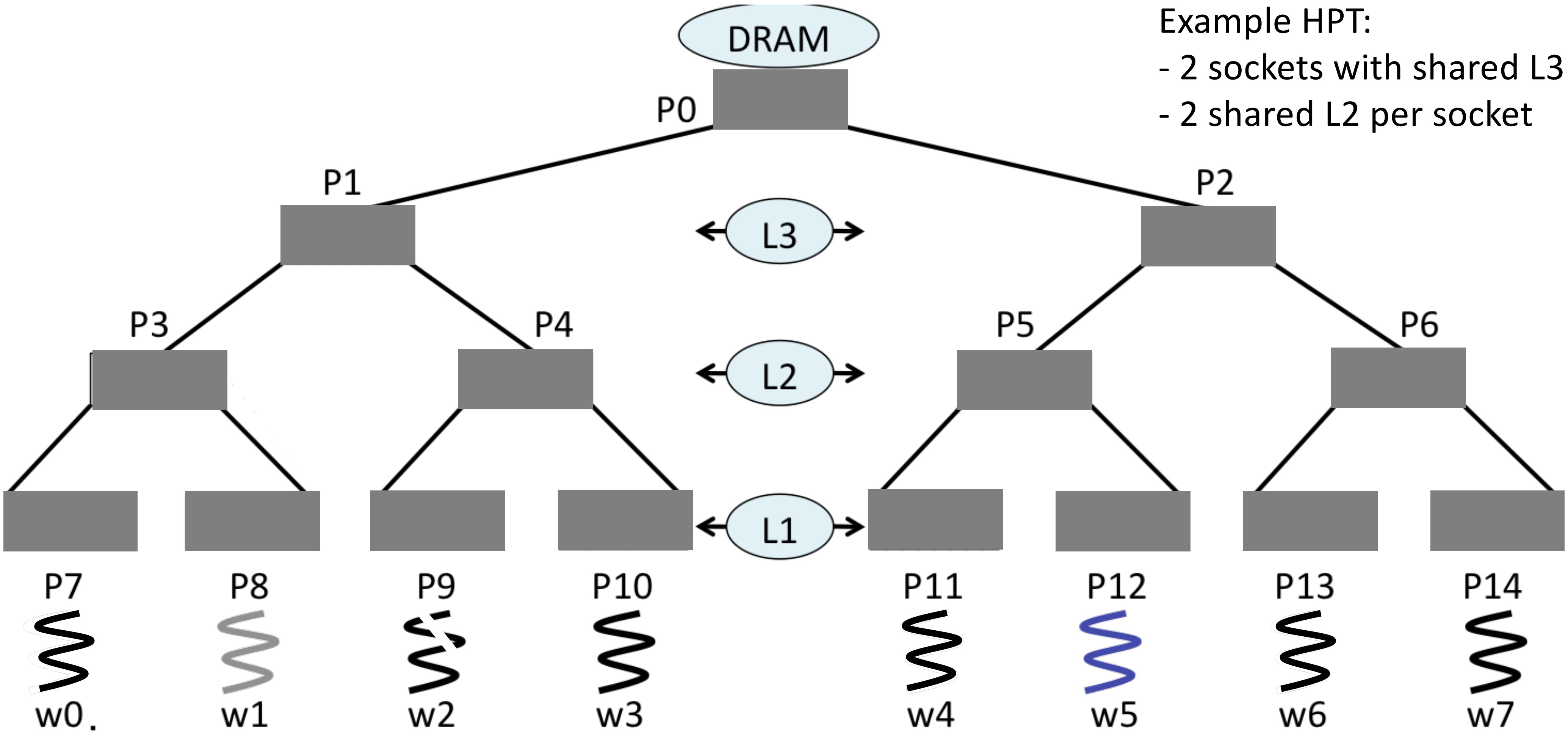
Speedup obtained with 24 threads over the sequential version



Dual socket 6 core Intel E5-2667 processor with hyperthreading. Array size 3MB and total iterations=100

Starting an Async at Non Leaf HPT Node?

Example HPT:
- 2 sockets with shared L3
- 2 shared L2 per socket

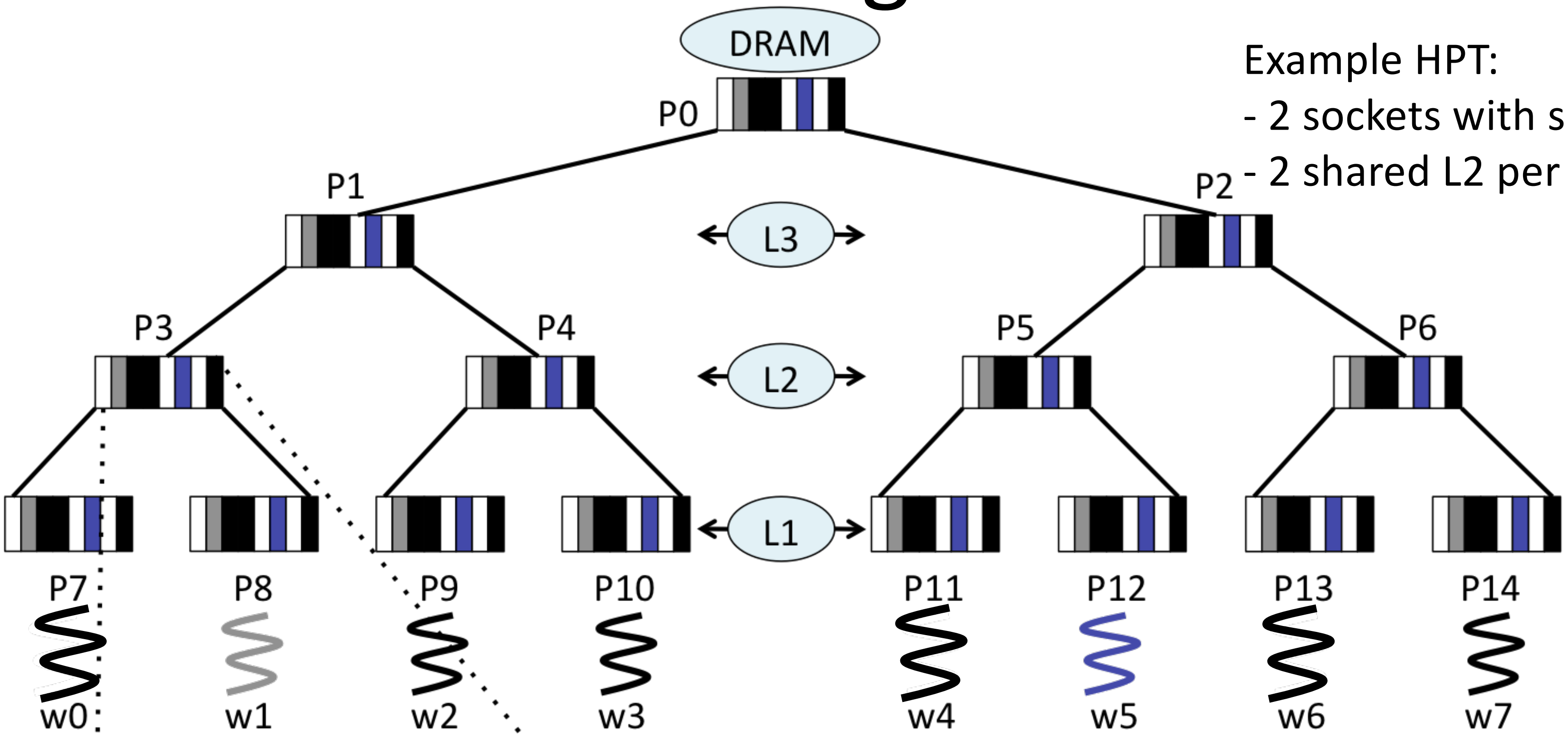


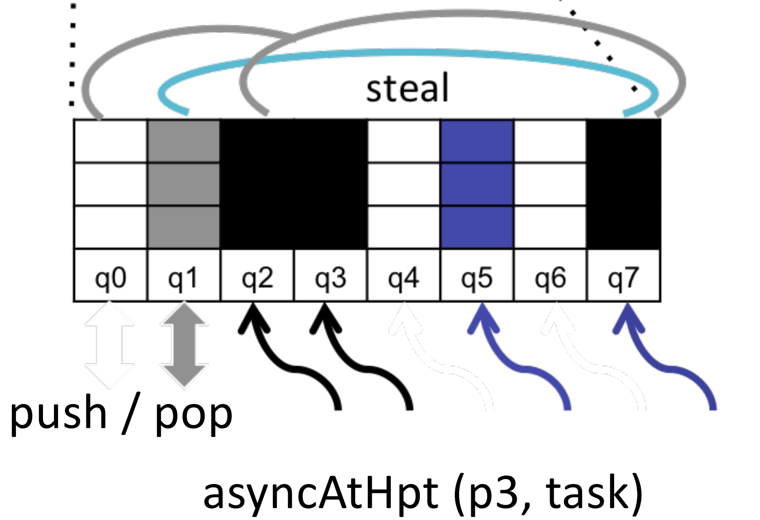
asyncAtHpt(P7, S1); asyncAtHpt(P9, S2); asyncAtHpt(P12, S3); asyncAtHpt(P14, S4);
asyncAtHpt(P2, S5);
asyncAtHpt(P4, S6);
asyncAtHpt(P6, S7);
asyncAtHpt(P0, S8);

Picture credit: Runtime Systems for Extreme Scale Platforms, PhD thesis, Sanjay Chatterjee, Rice University, 2013

Work-Stealing in a HPT

Example HPT:
- 2 sockets with shared L3
- 2 shared L2 per socket





- Workers attach to (own) leaf places
- Each place has one queue per worker
  - Ensures non-synchronized push and pop
- Any worker can push a task at any place
- Pop / steal access permitted to subtree workers
- Workers traverse path from leaf to root
- Tries to pop, then steal, at every place
- After successful pop / steal worker returns to leaf
- Worker threads are bound to cores

Picture credit: Runtime Systems for Extreme Scale Platforms, PhD thesis, Sanjay Chatterjee, Rice University, 2013

Next Class (Tomorrow)

- Promises, futures, and data driven tasks

Reading Material

- Hierarchical Place Trees: a Portable Abstraction for Task Parallelism and Data Movement, Yan et. al., LCPC 2009
  - http://www.cs.rice.edu/~vs3/PDF/hpt.pdf

Acknowledgements

- Several of the slides used in this course are borrowed from the following online course materials:
  - Course COMP322, Prof. Vivek Sarkar, Rice University
  - Course COMP 422, Prof. John Mellor-Crummey, Rice University
  - Course CSE539S, Prof. I-Ting Angelina Lee, Washington University in St. Louis
- Contents are also borrowed from following sources:
  - "Introduction to Parallel Computing" by Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. Addison Wesley, 2003
  - https://computing.llnl.gov/tutorials/parallel_comp/
  - https://images.google.com/