

Programming multicore systems using hierarchically tiled arrays

Diego Andrade and **Basilio B. Fraguera**
Computer Architecture Group (*HIPEAC institution*)
University of A Coruña (Spain)

James Brodman and **David Padua**
Polaris Group
University of Illinois
at Urbana-Champaign (USA)



Introduction

Hierarchically Tiled Array (HTA)

- Data type (class) with operations
- Array divided into tiles. Each tile can be either a regular array or an HTA (recursivity)
- Tiles express (data) parallelism and locality

D. Andrade
J. Brodman
B.Fraguela
D. Padua

Programming multicore systems using HTAs



Polaris
Research Group



Introduction

Hierarchically Tiled Array (HTA)

- Single-threaded view of the execution
- Global view of the data in distributed environments
- There are C++ and Matlab™ implementations of the library
- Same implementation valid for distributed/shared/hybrid memory

D. Andrade
J. Brodman
B.Fraguela
D. Padua

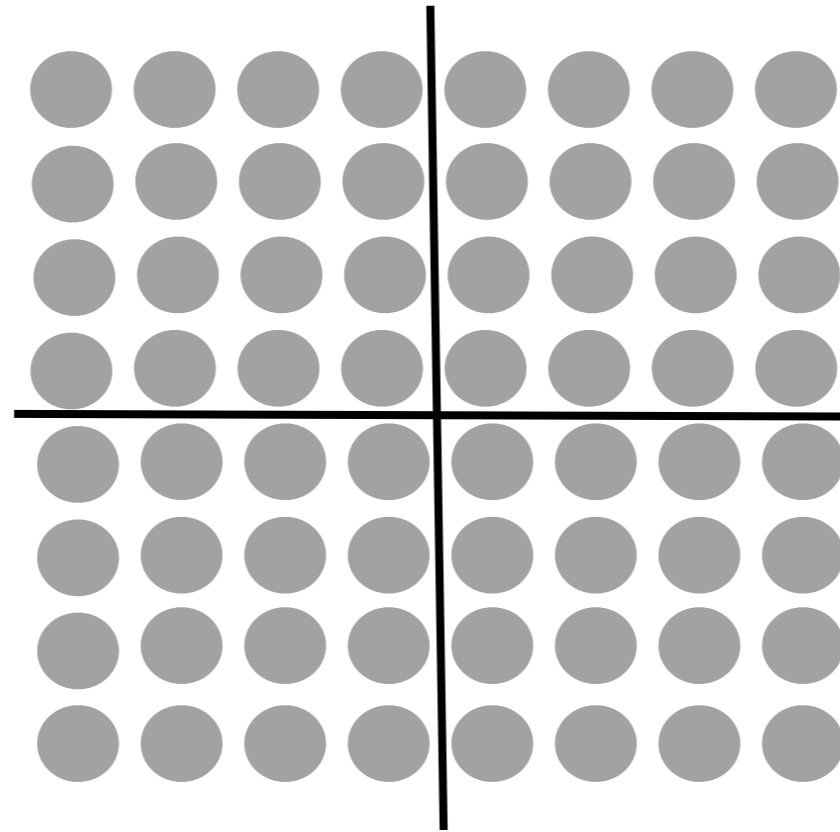
Programming multicore systems using HTAs



Polaris
Research Group



A (C++) HTA is born



```
Tuple<2>::Seq tiling=(Tuple<2>(4,4),Tuple<2>(2,2));  
HTA<int,2> h=HTA<int,2>::alloc(1,tiling,ROW);
```

Description of the tiling structure
Creation of the HTA

Type of each element
Number of dimensions

Levels of tiling
Memory Layout

D. Andrade
J. Brodman
B.Fraguela
D. Padua

Programming multicore systems using HTAs



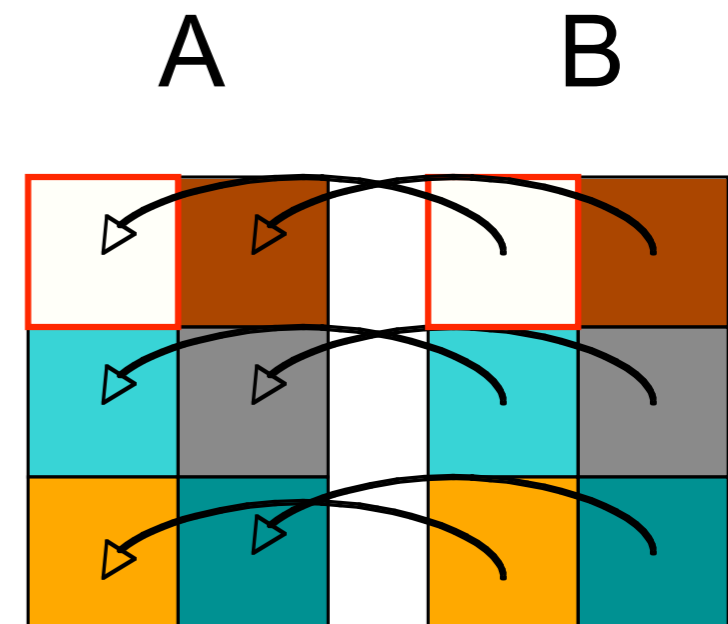
Polaris
Research Group



Data parallel operations

- Element-by-element:
 - Array notation: $A = A + \alpha * B$
 - `hmap(funcutor)`
- Reductions: `reduce(funcutor)`
- Scans: `scan(funcutor)`

```
struct Example {  
    operator() (HTA A, HTA B) {  
        A = A + alpha * B;  
    }  
};  
  
A.hmap(Example(), B);
```



D. Andrade
J. Brodman
B. Fraguera
D. Padua

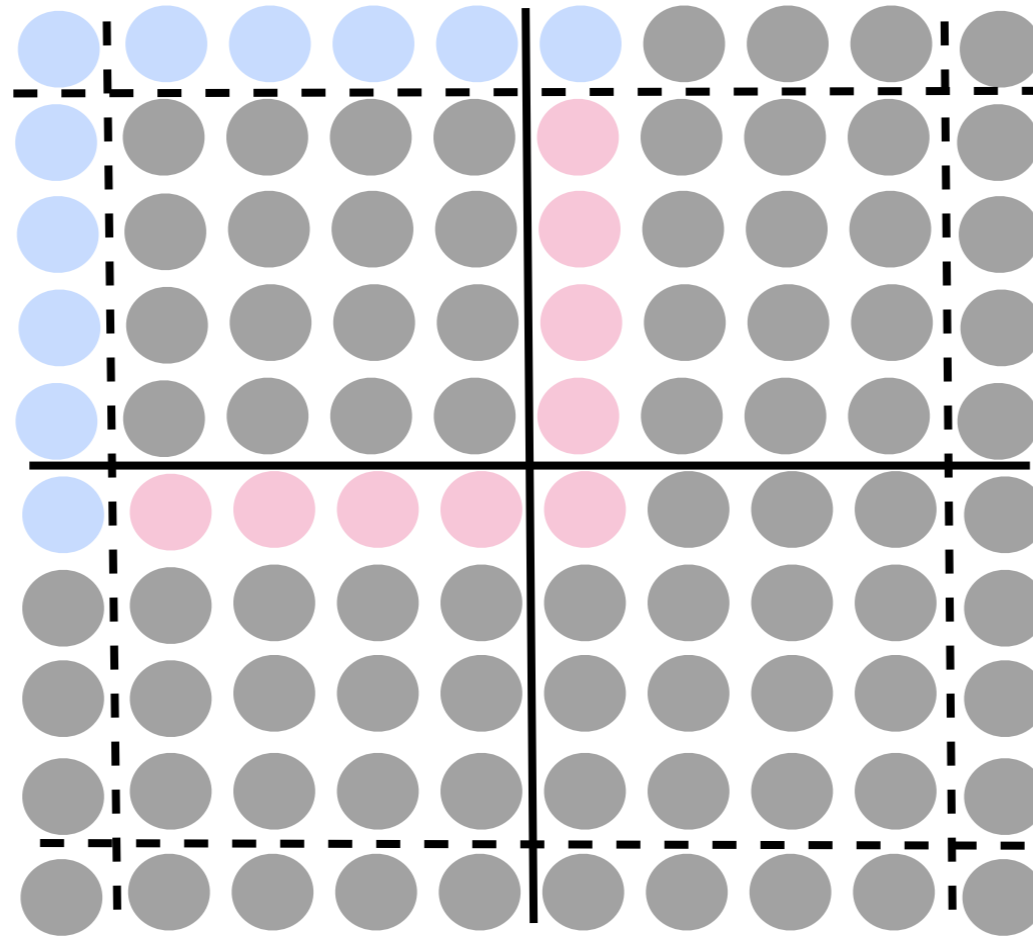
Programming multicore systems using HTAs



Polaris
Research Group



Extra useful features: overlapped tiling



```
Tuple<2>::Seq tiling=(Tuple<2>(4,4),Tuple<2>(2,2));
```

```
Overlap<2> ol(Tuple<2>(1,1),Tuple<2>(1,1));
```

```
HTA<int,2> h=HTA<int,2>::alloc(1,tiling,ol,ROW);
```

Object specifying overlap between tiles

Overlap in **negative** direction in each dimension

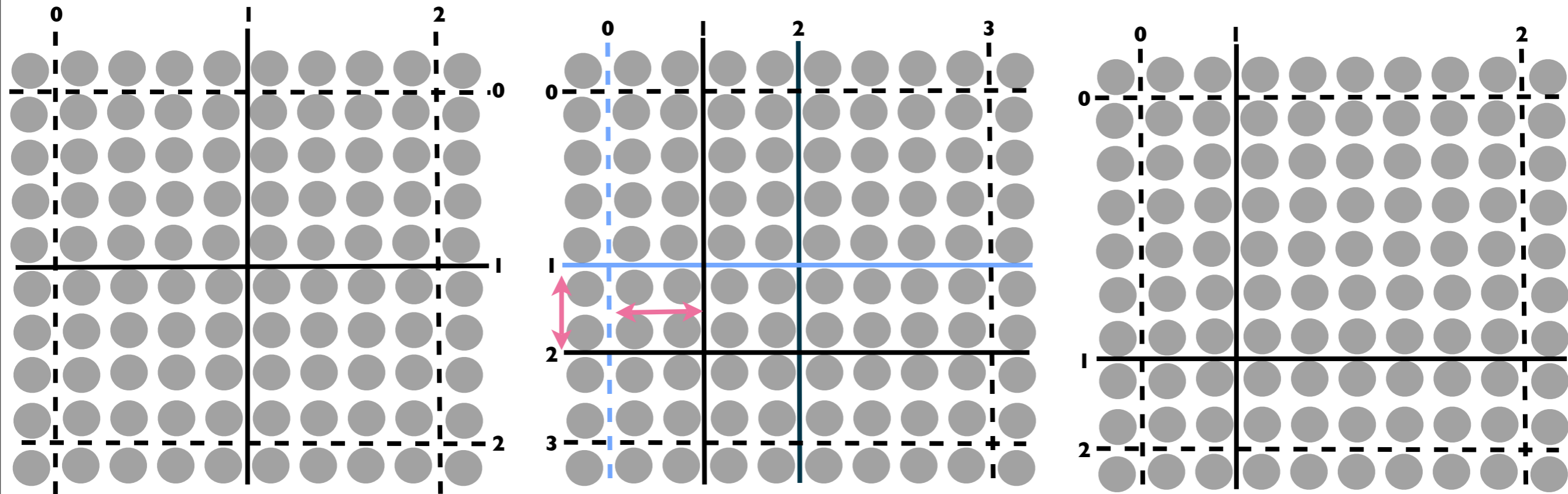
Overlap in **positive** direction in each dimension

D. Andrade
J. Brodman
B.Fraguela
D. Padua

Programming multicore systems using HTAs



Extra useful features: dynamic partitioning



```
h.part(Tuple<2>(1,0), Tuple<2>(2,2));
```

```
h.rmpart(Tuple<2>(1,2));
```

Partition lines relative to which the new ones will be positioned

Offset in elements for new partition lines

D. Andrade
J. Brodman
B.Fraguela
D. Padua

Programming multicore systems using HTAs



Programming using HTAs

Programmability is difficult to measure

- A metric: *Source lines of code*
- Implementation of some kernels:

- Extracted from <http://softwarecommunity.intel.com/articles/eng/1359.htm>

Kernel examples of the Intel© Thread Building Blocks book

D. Andrade
J. Brodman
B. Fraguera
D. Padua

Programming multicore systems using HTAs



Polaris
Research Group



Average: sequential version

```
for(i=0;i<N;i++) {  
    output[i]=(input[i-1]+ input[i] input[i+1])*(1/3.0f);  
}
```



Average: HTA version

```
typedef HTA<float,1> HTA_1;
#define T1(i) Tuple<1>(i);

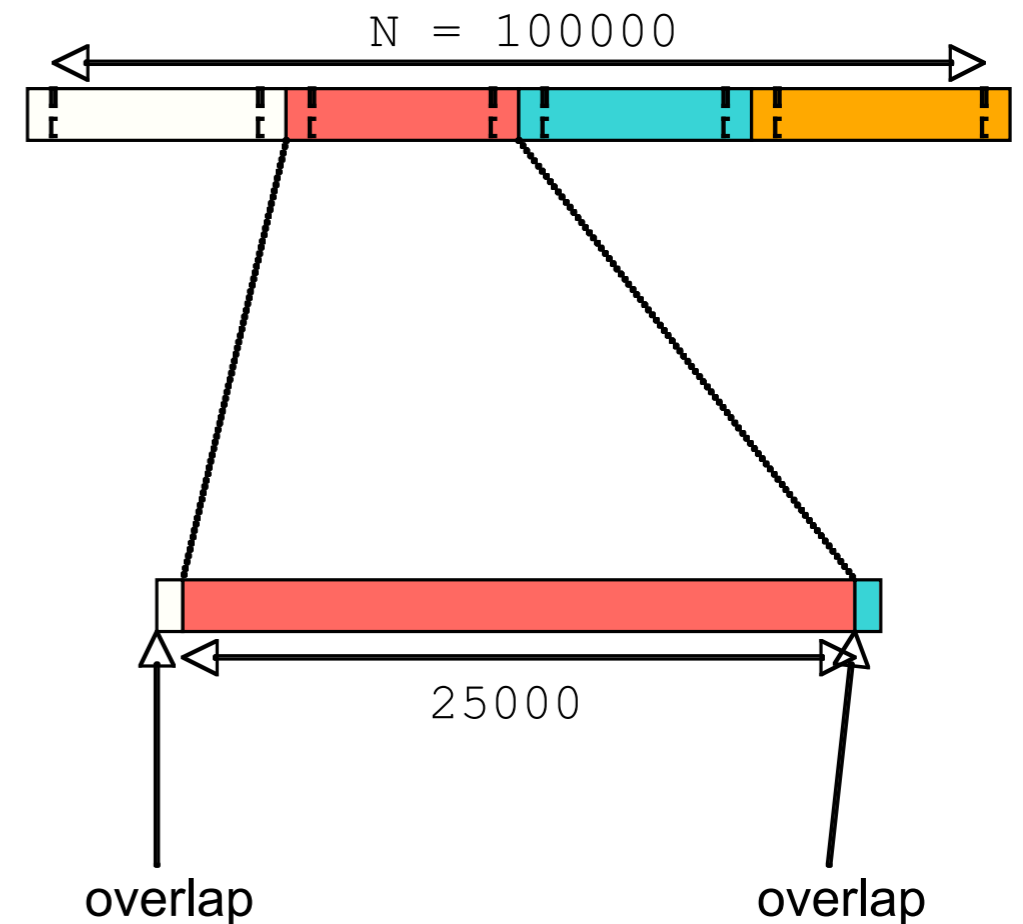
struct Average {
  void operator()(HTA_1 input, HTA_1 output) const {
    for( int i = 0; i != input.shape().size()[0]; ++i )
      output[i] = (input[i-1]+input[i]+input[i+1])*(1/3.0f);
  }
};

const int N = 100000;
static int nTiles = 4;

int main( int argc, char* argv[] ) {
  Traits::Default::init(argc,argv);

  Seq< Tuple<1> > tiling(T1(N/nTiles),T1(nTiles));
  Overlap ol(T1(1),T1(1));
  HTA_1 input = HTA_1::alloc(1,tiling,ol,NULL,ROW);
  HTA_1 output = HTA_1::alloc(1,tiling,NULL,ROW);
  ... /* Initialization not shown */

  input.hmap(Average(), output);
  return 0;
}
```



D. Andrade
J. Brodman
B. Fraguera
D. Padua

Programming multicore systems using HTAs



Seismic: Array operations

```
M[1:0.3*UH] = 1.0/32;  
Material[1:0.3*UH] = WATER;  
M[0.3*UH+1:0.5*UH] = 1.0/8;  
Material[0.3*UH+1:0.5*UH] = SANDSTONE;  
M[0.5*UH+1:0.7*UH] = 1.0/2;  
Material[0.5*UH+1:0.7*UH] = SHALE;  
M[0.7*UH+1:UH-1] = 1.0/8;  
Material[0.7*UH+1:UH-1] = SANDSTONE;
```

D. Andrade
J. Brodman
B.Fraguela
D. Padua

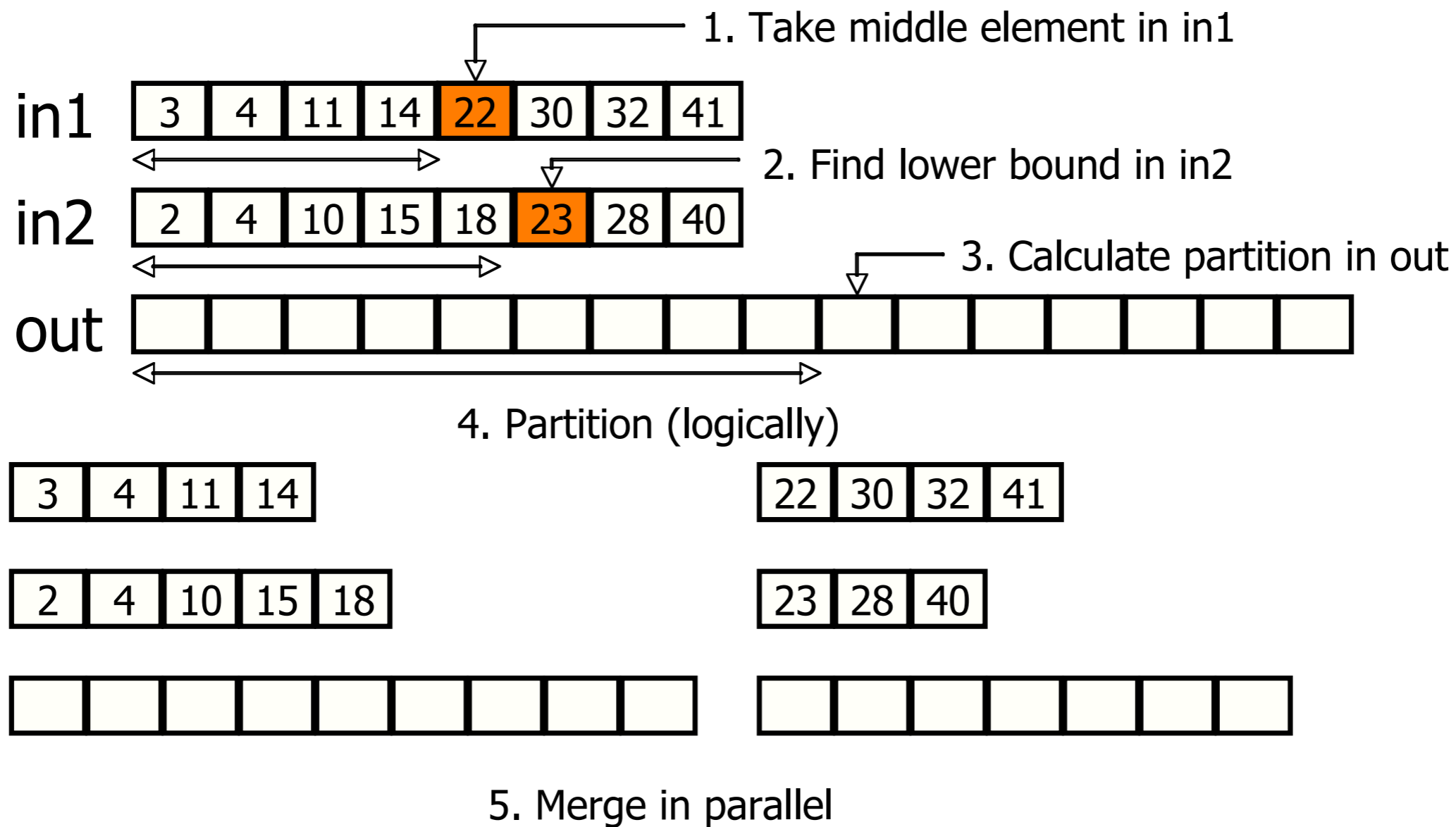
Programming multicore systems using HTAs



Polaris
Research Group



Parallel merge: Dynamic/irregular parallelism



D. Andrade
 J. Brodman
 B.Fraguela
 D. Padua

Programming multicore systems using HTAs



Parallel Merge: HTA code

```
struct PMerge {
  void operator() (HTA1 out, HTA1 in1, HTA1 in2) {
    int in1_size = in1.lsize(0);
    if (in1_size > GRAINSIZE) { /* parallel merge */
      int midpos_in1 = in1_size / 2 ;
      int cutpoint_in2 = in2.lowerBoundPos(in1[midpos_in1]);
      in1.part( [0], [midpos_in1] );
      in2.part( [0], [cutpoint_in2] );
      out.part( [0], [midpos_in1 + cutpoint_in2] );
      out.hmap(PMerge(), in1, in2);
      in1.rmPart();
      in2.rmPart();
      out.rmPart();
    } else {
      /* sequential merge */
    }
  }
};

...
out.hmap(Pmerge(), in1, in2);
```

D. Andrade
J. Brodman
B.Fraguela
D. Padua

Programming multicore systems using HTAs



Polaris
Research Group



HTAs vs Intel© TBBs comparison

Comparison of the HTA(C++) library and the Intel© Thread Building Blocks library from two points of view:

- Programmability
- Performance

Intel© TBB is a C++ library that defines tasks and how to partition them

D. Andrade
J. Brodman
B.Fraguela
D. Padua

Programming multicore systems using HTAs



Polaris
Research Group



SLOC comparison

Code	SLOC HTA	SLOC TBB	Reduction
Average	28	39	28%
Seismic	304	295	-3%
Parallel merge	70	74	54%
Game of life	97	309	69%
Substring finder	49	49	0%

D. Andrade
J. Brodman
B. Fraguera
D. Padua

Programming multicore systems using HTAs



Polaris
Research Group



Performance comparison

Code	Time (in milliseconds)									
	HTA					TBB				
	1	2	3	4	8	1	2	3	4	8
Average	490	403	381	260	253	536	193	189	190	196
Seismic	1993	1060	1010	778	503	1500	802	832	670	483
Parallel merge	8783	4704	4591	3885	3365	11823	5543	5144	3968	3793
Game of life	21472	115731	8568	6802	5182	19788	11685	8976	7593	5520
Substring finder	6180	3130	2350	1570	810	6413	3200	2130	1605	810

Quad-core 2.66 Ghz Xeon processors

gcc 4.2.1 -03

D. Andrade
J. Brodman
B.Fraguela
D. Padua

Programming multicore systems using HTAs



Control of granularity

- Previous experiments created “one tile per core” in the HTA
- Alternatively, we can create more tasks than processors:
 - Benefit from better load balance, dynamic distribution.
 - E.g. Game of Life: 25% speedup on TBB version.

Conclusions

- HTA is a good alternative to improve programmability of multicore machines
- It achieves a performance similar to a commercial alternative like Intel© Thread Building Blocks
- It can coexist in the same code with other libraries/ extensions like Intel© TBB, Posix Threads, OpenMP...

**D. Andrade
J. Brodman
B.Fraguela
D. Padua**

Programming multicore systems using HTAs



Polaris
Research Group



Questions?

More info at <http://polaris.cs.uiuc.edu/hta>

D. Andrade
J. Brodman
B. Fraguera
D. Padua

Programming multicore systems using HTAs



Polaris
Research Group

