Design and Usage of htalib – a C++ Library for Hierarchically Tiled Arrays

Ganesh Bikshandi, Jia Guo, Christoph von Praun*, Gabriel Tanase**, Basilio. B. Fraguela***, Maria J. Garzaran, David Padua and Lawrence Rauchwerger**

University of Illinois, Urbana-Champaign, IL

*IBM T. J. Watson Research Center, Yorktown Heights, NY

**Texas A&M University, College Station, TX

***University da Coruna, Spain

Outline

- Motivation ←
- HTA & Prior results
- Design of htalib & new features
- Experimental results
- Conclusion & Future work

Motivation

- Combining CPUs in future for speed
  - Multicores, parallel & distributed systems
- But, efficient parallel programming is extremely difficult

Motivation

- MPI is the current state-of-the-art for distributed memory model
  - SPMD + Local view + Multi-threaded model
- Offers performance but productivity is hampered
  - Complex bugs (deadlocks, races)
  - Poor mapping between algorithm and the final code
- Often equated to assembly language.

New model for parallel programming

Hierarchically Tiled Arrays (HTA)

- Sequential, determinate logic for the programmer
- "Global" shared memory view

Hierarchical tiling is explicit and translates to ...

- Parallelism and data distribution (productivity)
- Locality of access (performance)

Key data structure: HTA

- Organization of data
- Operations

- Initially implemented in MATLAB

This talk – a C++ library for HTA (htalib)

- C++
  - Offers several performance benefits over MATLAB
  - Compilers perform aggressive scalar optimizations
- Introduce new HTA operations
  - map, reduce, mapReduce, overlapped tiling & data layering

more performance + more productivity

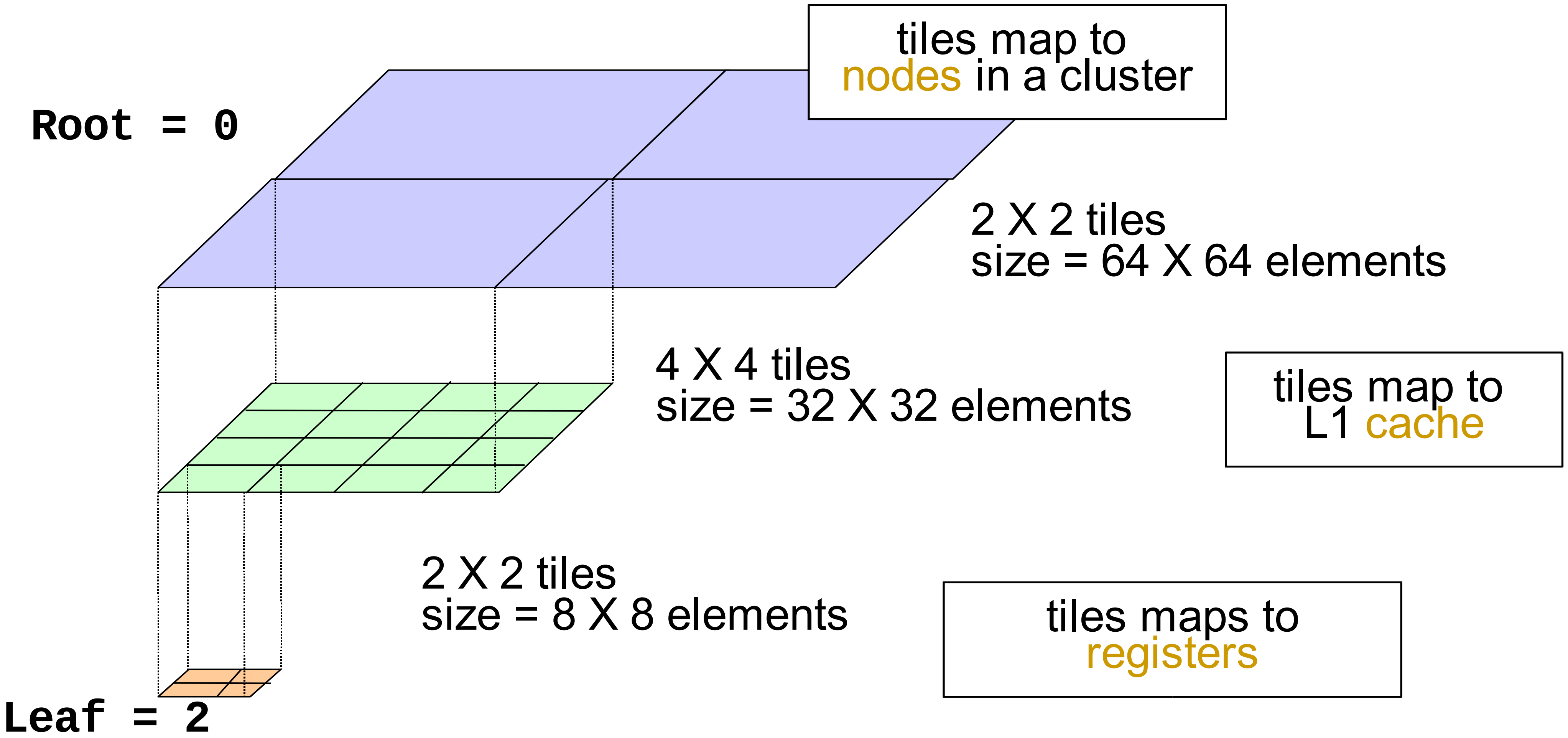
Outline

- Motivation
- HTA & Prior results ←
- Design of htalib & new features
- Experimental results
- Conclusion & Future work

Hierarchically Tiled Arrays

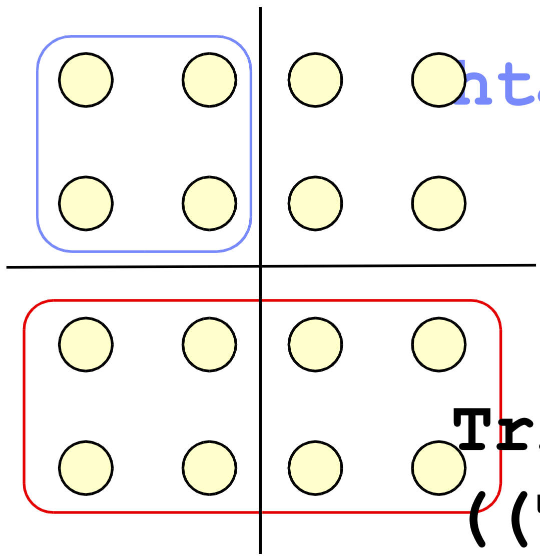
- Treats tiles as first class data types
- HTA
  - Is a recursive data type (arrays of arrays of arrays..)
  - Support block recursive operations
  - Provides global view and single threaded model
  - Suitable for array computations

Hierarchically Tiled Arrays



Tile access

hta



(I) `hta (Tuple (0,1))`

(II)

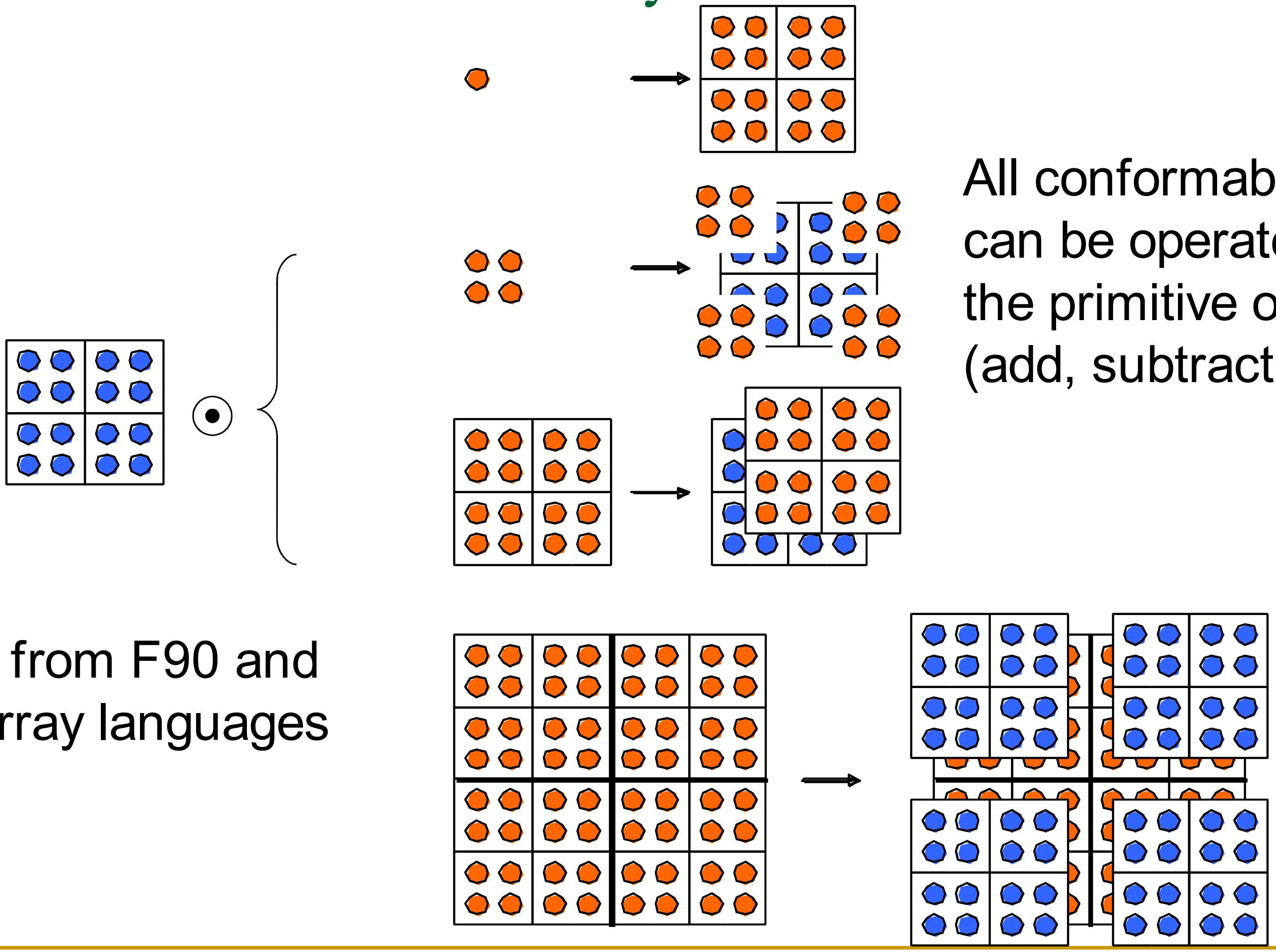
```
Triplet::Seq selection =
((Triplet(1,1), Triplet(0,1)));
hta (selection)
```

Syntax short form:

(I) `hta (0,1)`

(II) `hta (1, 0:1)`

HTA Conformability

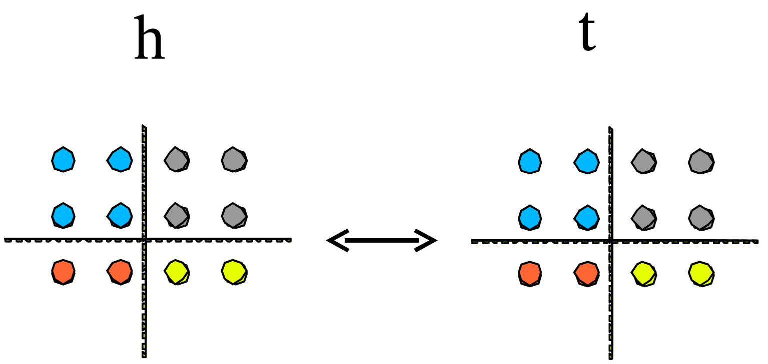


All conformable HTAs can be operated using the primitive operations (add, subtract, etc)

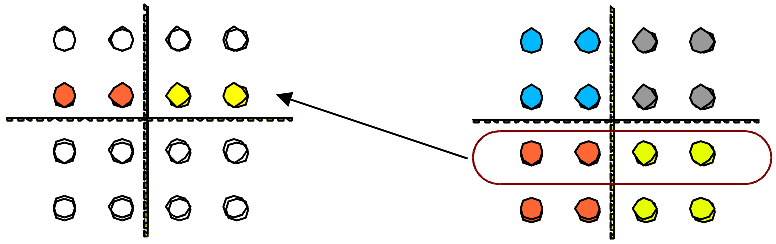
Inspired from F90 and similar array languages

HTA Assignment

h(:,:) = t(:,:)

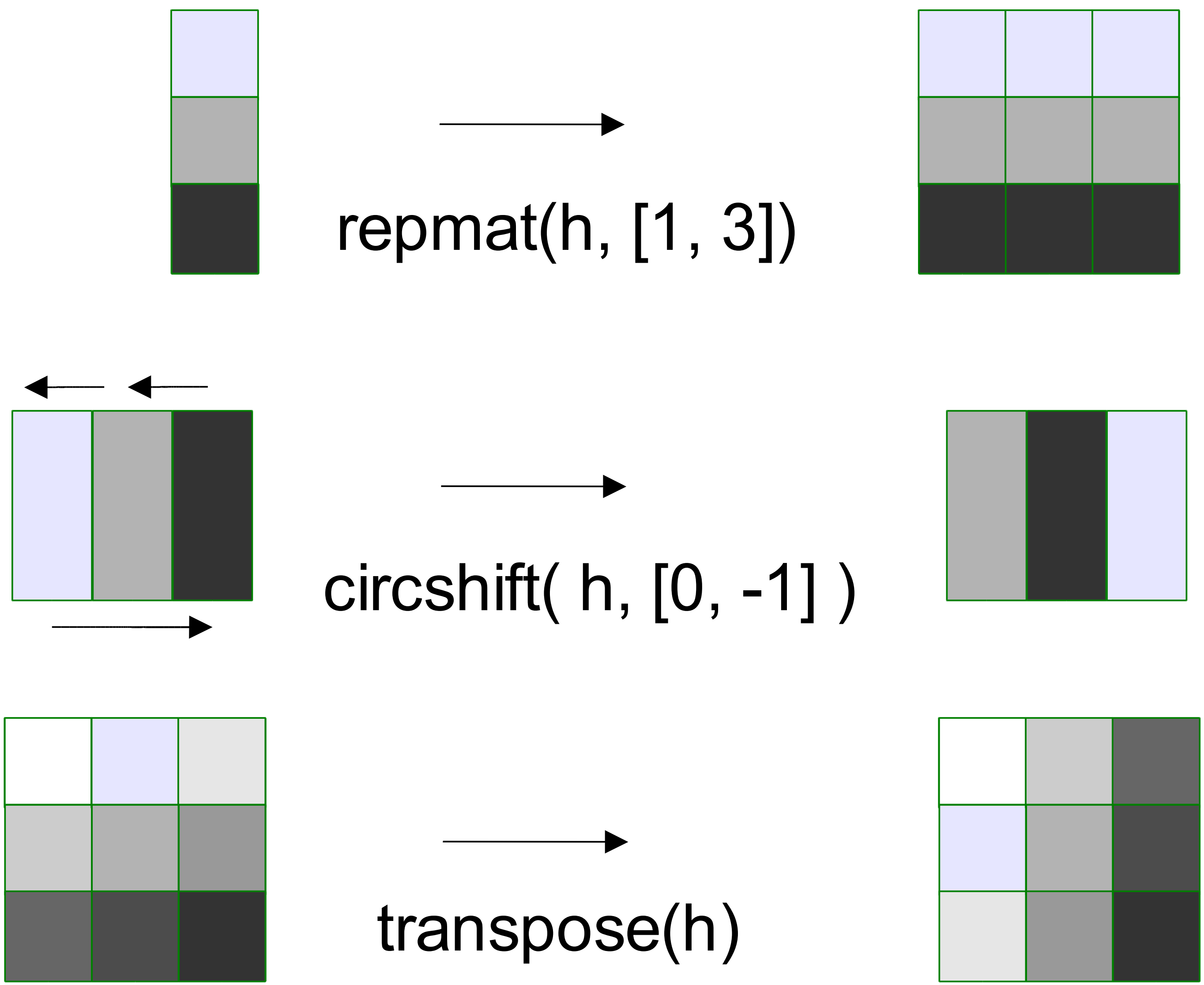


h(0,:)[2,:] = t(1,:)[0,:]



Explicit communication

Higher level HTA operations



HTA example: SUMMA Matrix Multiplication

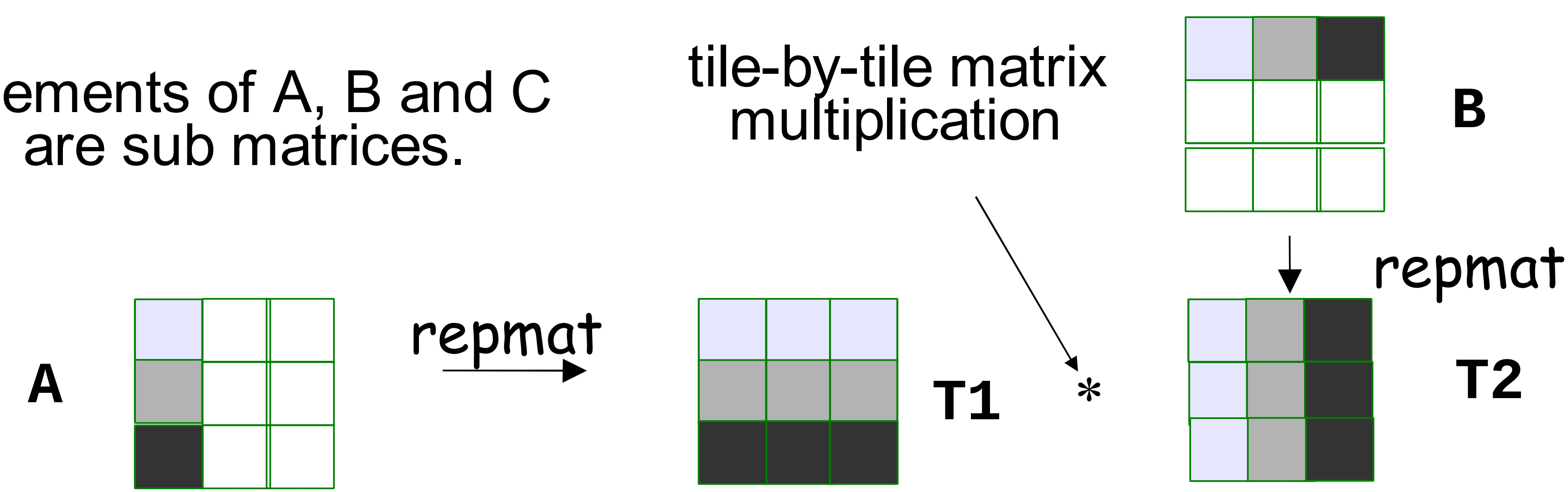
```
for  k=1:n
          C(:,:)=C(:, :)+A(:, k) * B(k, :)
end
```

Rank 1 update

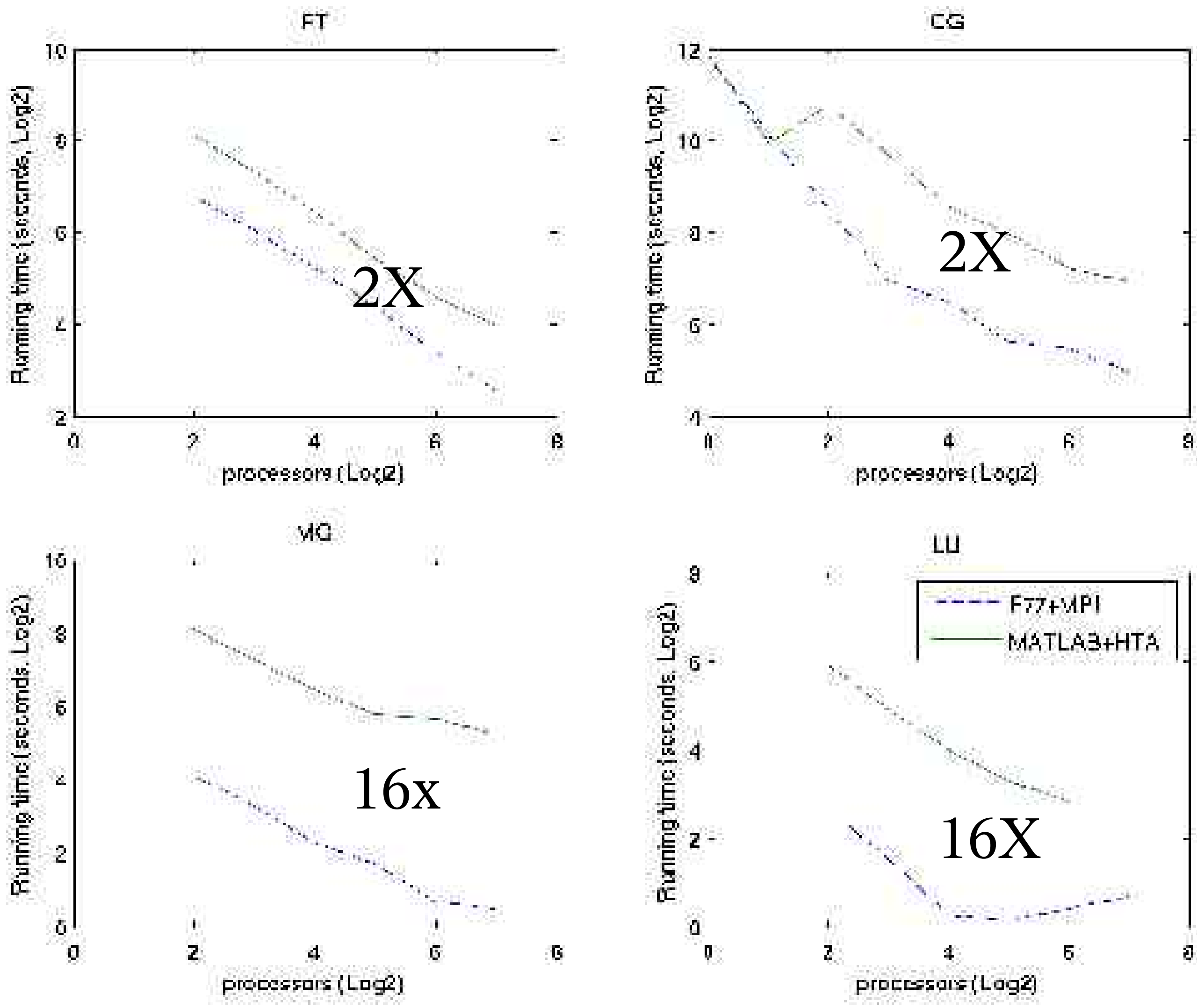
outer product MMM

```
void summa (A, B, C) {
     for (int k = 0; k < m; i++) {
          T1 = repmat(A(:, k), 1, m);
          T2 = repmat(B(k, :), m, 1);
          C = C + T1 * T2;
     }
}
```

Parallel MMM (HTA)

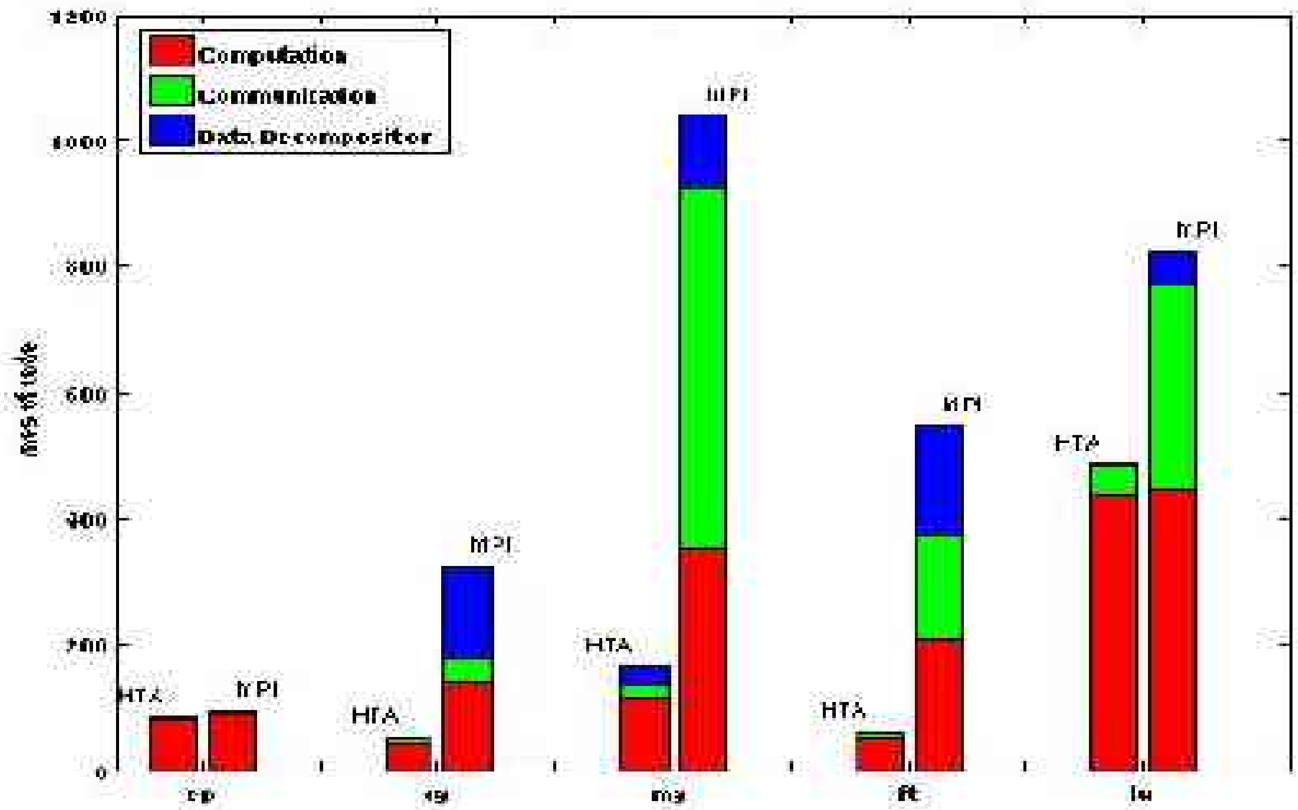


Earlier (MATLAB HTA library) results: Running time



Intel
Linux
Cluster

MATLAB results: LOC



Programming for Parallelism and Locality with Hierarchically Tiled Arrays, PPoPP06, NYC, USA, March 2006

Outline

- Motivation
- HTA & Prior results
- Design of htalib & new features ←
- Experimental results
- Conclusion & Future work

C++ implementation

- Designed for scaling
- Uses templates
  - Benefits from specialization and instantiation
- Generic design
  - Polymorphism
- Automated memory management
  - Reference Counting

Optimization (I)

- Lazy evaluation of binary operations
- Avoids temporaries during expression evaluation & redundant copying during assigment

```
a = b + c;  //no data dependence between lhs and rhs
       ↓
a = BinExpr (b, c, +);
       ↓
HTA operator = (BinExpr & e) {
   for (int i = 0; i < this->shape().card(); i++)
     this->data_[i] = arg1_.data_[i] + arg2_.data_[i];
}
```

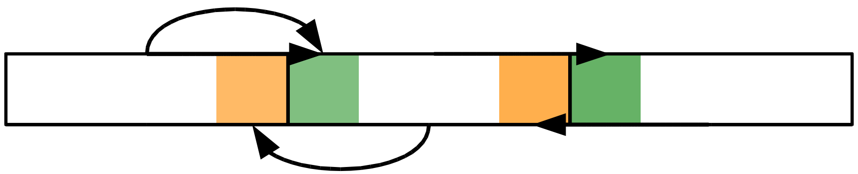
a += b + c; a -= c + d; a = b * c; a = b - c;
a = 0.5 * b; a = d / 2; a = b / c; a = c - d;

Optimization (II)

- Asynchronous overlap

```
(Earlier implementation)
B(1:n)[0] = B(0:n-1)][d];
(barrier)
B(0:n-1)[d+1] = B(1:n)[1];
(barrier)

htalib::async();
B(1:n)[0] = B(0:n-1)][d];
(no barrier)
B(0:n-1)[d+1] = B(1:n)[1];
htalib:sync(); (barrier)
```



- Condition
  - No dependence within enclosing async and sync.

Operator framework

Generalization of our MATLAB library
- Addition of level argument (to control the level of application)

Primitive operators
- with scalar arguments (+, -, max, ...)
- with tiles as arguments (+, -, permute, matmul, ..)

High-level operators
- map (unordered), do-all parallelism
- scan (ordered), pipeline parallelism
- Reduction
- map-reduce, (divide and conquer)

Scope of operation
- a single level (tiles or scalars)
- recursive application to multiple-level

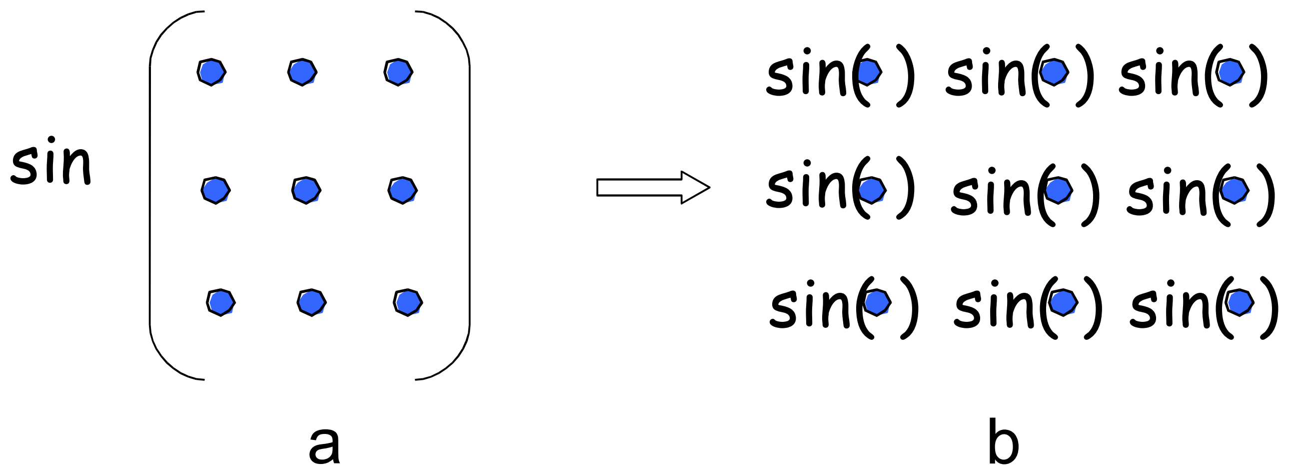
Primitive operators

- STL-like functor objects

```
struct plus {
   double operator () (const double a, const double b {
       return a + b;
     }
   };

struct ft {
   void operator() (Array* x) {
      //...
      FFTforward (...);
   }
}
```

Data Parallel Array Functions (Maps)



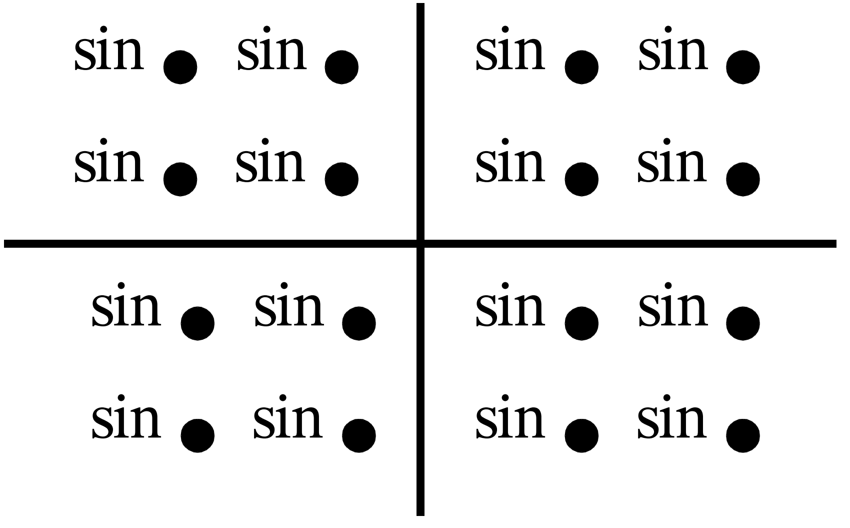
HTA Map

map (op, [rlevel])

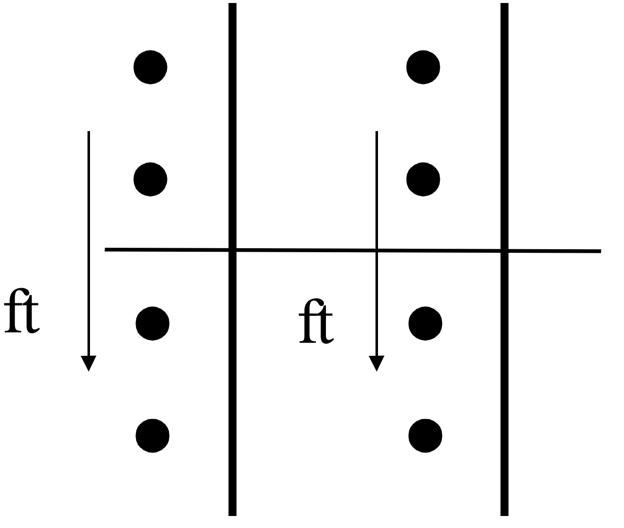
- op = any scalar or array or HTA or Higher level operation
- rlevel = termination level of recursion

(default = scalars)

Map



`h.map(sin())`



`h.map(ft(), 1)`

Reduction Operations

Reduction: An operation applied to all the components of a vector to produce a scalar

```
reduce (+, [1, 9, 13]) ⟹ 23
```

In general, an operation applied to all the components of a n-dimensional array to produce a n-1 dimensional array.

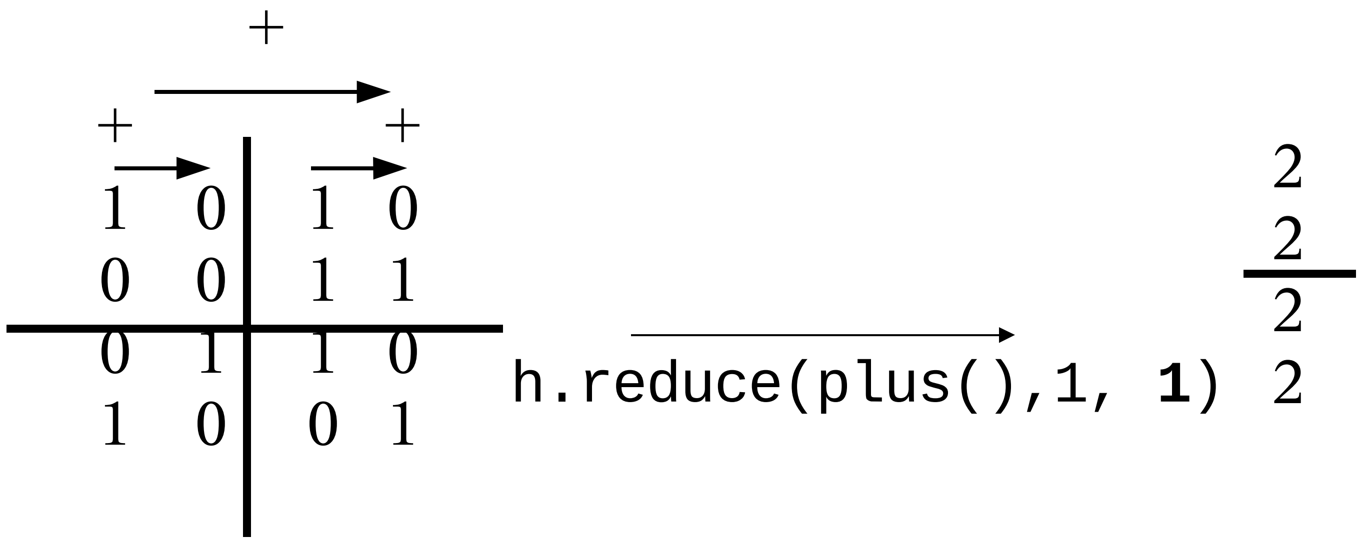
For matrices, row reduction and column reduction

HTA Reduce

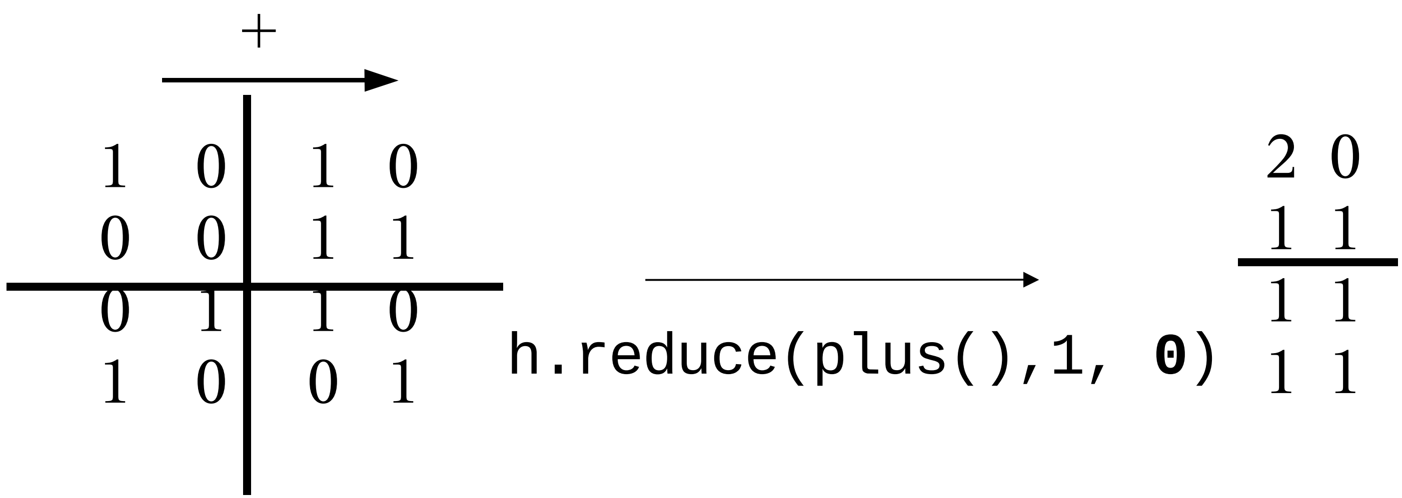
reduce(op, dim, [rlevel])

- op – any associative and commutative operation
- dim – dimension of reduction
- rlevel – termination level of recursion

HTA Reduce



+
+ +
1 0 1 0
0 0 1 1
0 1 1 0
1 0 0 1
h.reduce(plus(),1, **1**)
2
2
2
2



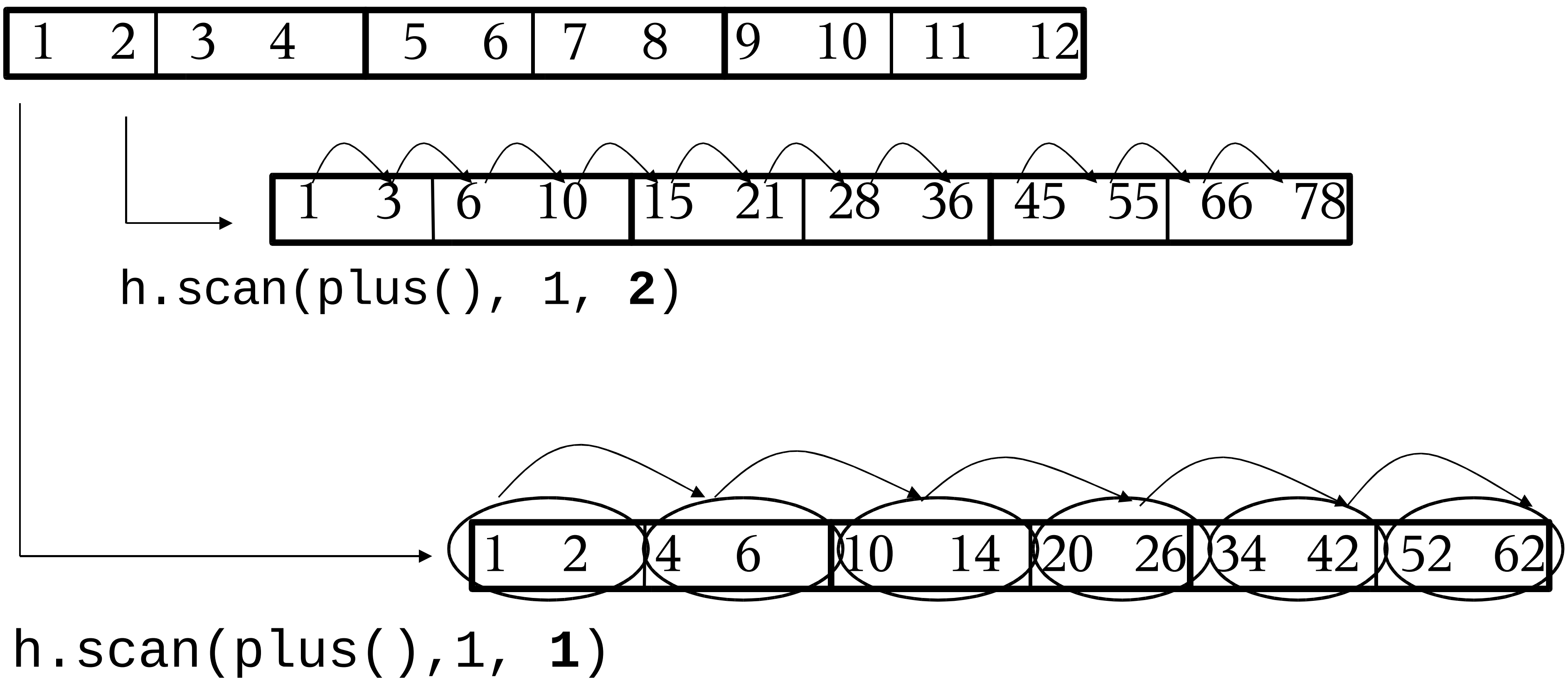
+
1 0 1 0
0 0 1 1
0 1 1 0
1 0 0 1
h.reduce(plus(),1, **0**)
2 0
1 1
1 1
1 1

HTA Scan

Scan (op, dim, [rlevel])

- op – any primitive operation
- dim – dimension of scan.
- rlevel – termination level of recursion

HTA Scan

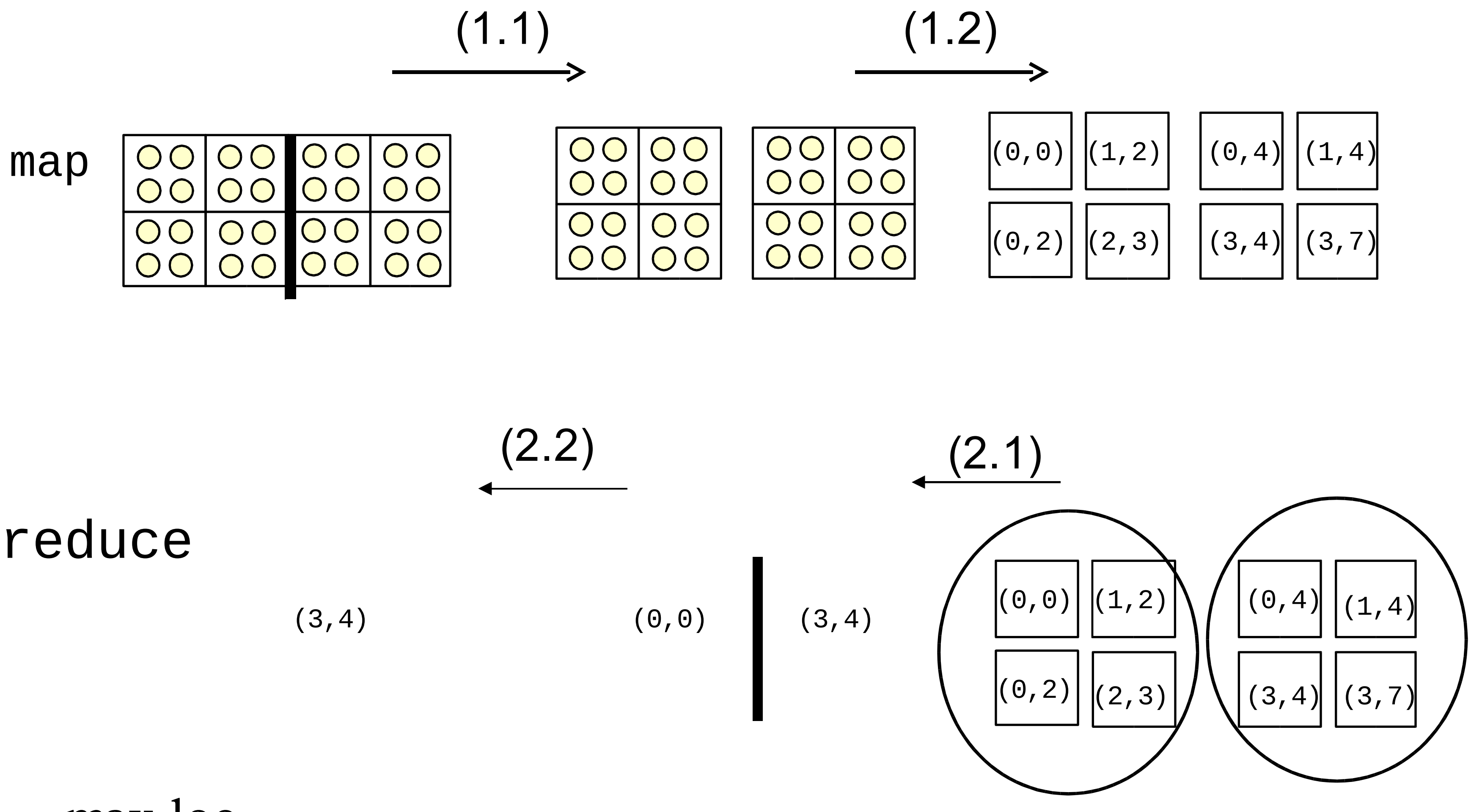


1 2 3 4 5 6 7 8 9 10 11 12
1 3 6 10 15 21 28 36 45 55 66 78
h.scan(plus(), 1, **2**)
1 2 4 6 10 14 20 26 34 42 52 62
h.scan(plus(),1, **1**)

Map-Reduce

- Framework for composing map and reduce
  - Example: Determine the maximum elements in HTA , together with their leaf-coordinates
- Sequential programming model
- Parallelism, communication, synchronization are implicit

Map-Reduce

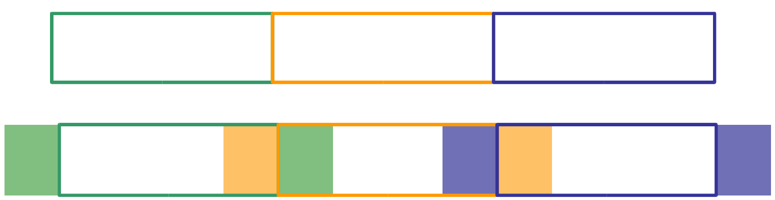


max-loc

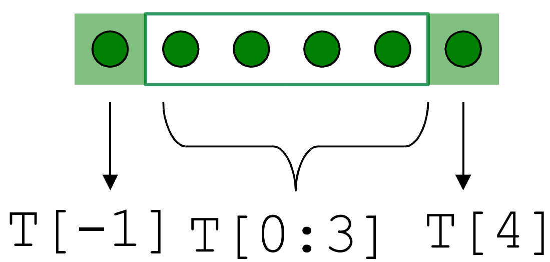
Overlapped tiling

- Shadow regions in HTA can be accessed across tiles.

```
Overlap<DIM> ol(1,1)
```



```
B(1:n)[0]=B(0:n-1)[d];
B(0:n-1)[d+1]=B(1:n)[1];
A()[1:d]=S*(B()[2:d+1]
            +B()[0:d-1]);
```

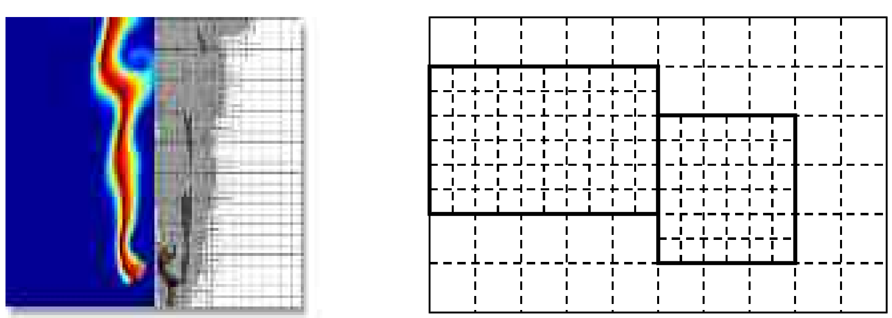



```
A=HTA<double,1>alloc(tiling,array,
ROW, ol);
```

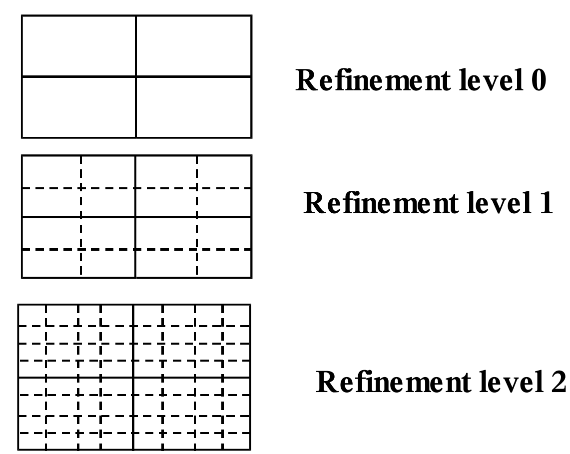
```
A()[0:3]=S*(B()[1:4]+B()[-1:2]);
```

Support for Multigrid Applications

- HTAs facilitate
  - ✓ Parallelism and data distribution
  - ✓ Locality of access
  - NEW: Implicit Support for Hierarchical Applications (productivity)
- Goal: support multi-grid applications
  - Model physical phenomena using finite-difference methods
  - Grid data structure used to discretize the continuum domain
  - HTA used to represent the hierarchical grid



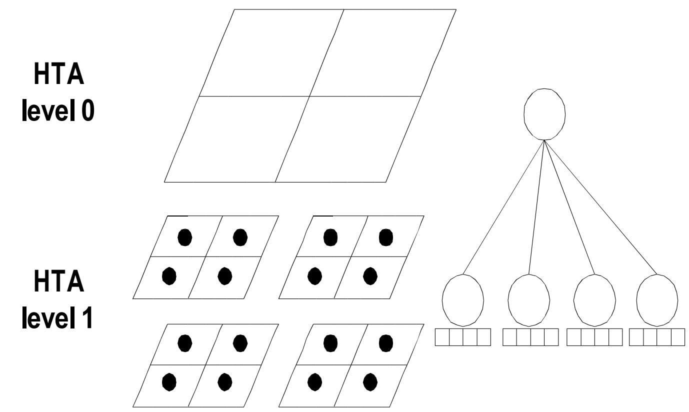
AMR



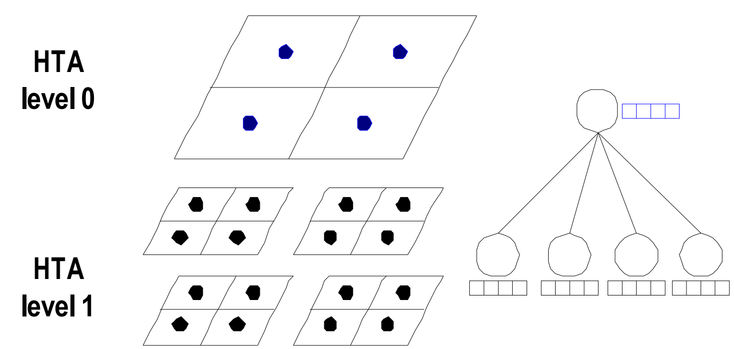
MG benchmark

HTA Extensions – Data Layering

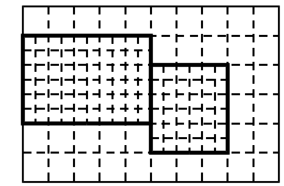
- HTAs
  - Leaf tiles: data
  - Other tiles: meta data to control data distribution and data locality



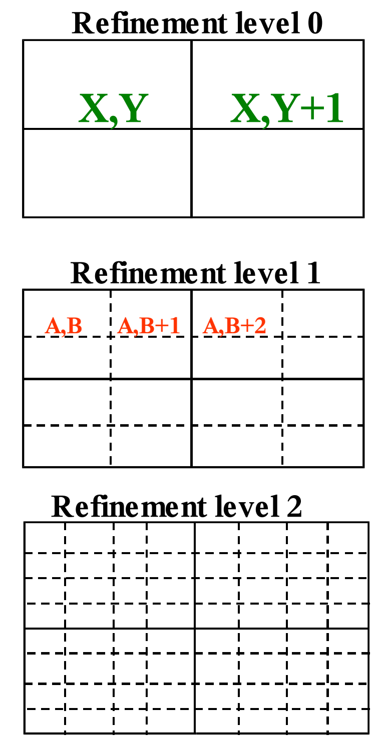
- HTAs with data layering
  - Data and meta-data at all levels
  - Levels correspond to different degrees of refinement in a multigrid application



HTA interface extensions for AMR



- void refine(level,region, refinement_factor)
- <region_coarse,region_fine> = project(level)
  - Compute how elements on two adjacent levels correspond to each other
- HTA get_level(level)
  - Converts a layer of a multi layered HTA to a regular HTA



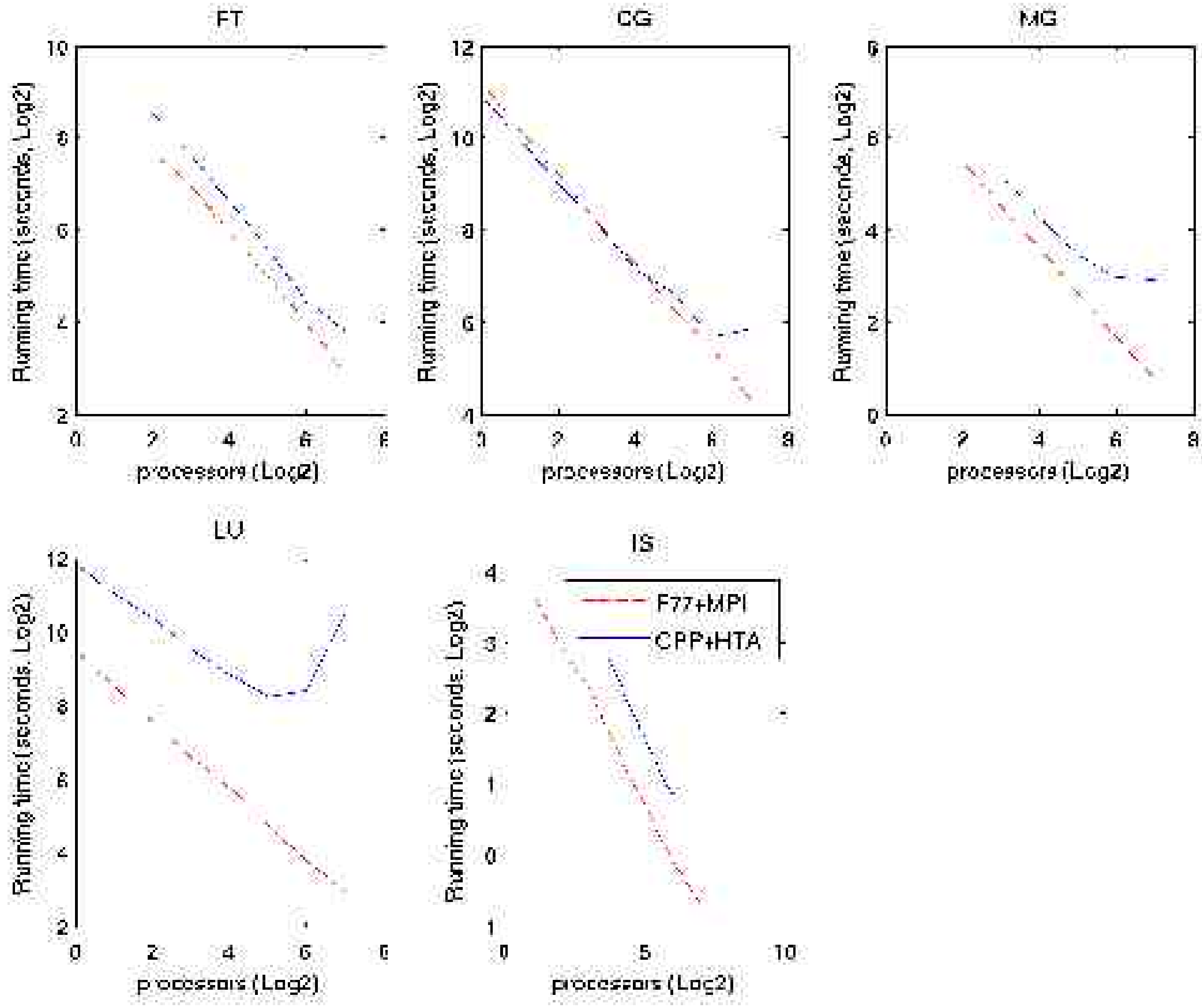
Example

```
HTA h = HTA::alloc(Tuple(2,2));
H.refine(0, ALL, 2);
H.refine(1, ALL, 2);
pair<region_coarse, region_fine> = H.project(level);
h.get_level(1)(region_fine) = h.get_level(0)(region_coarse);
```

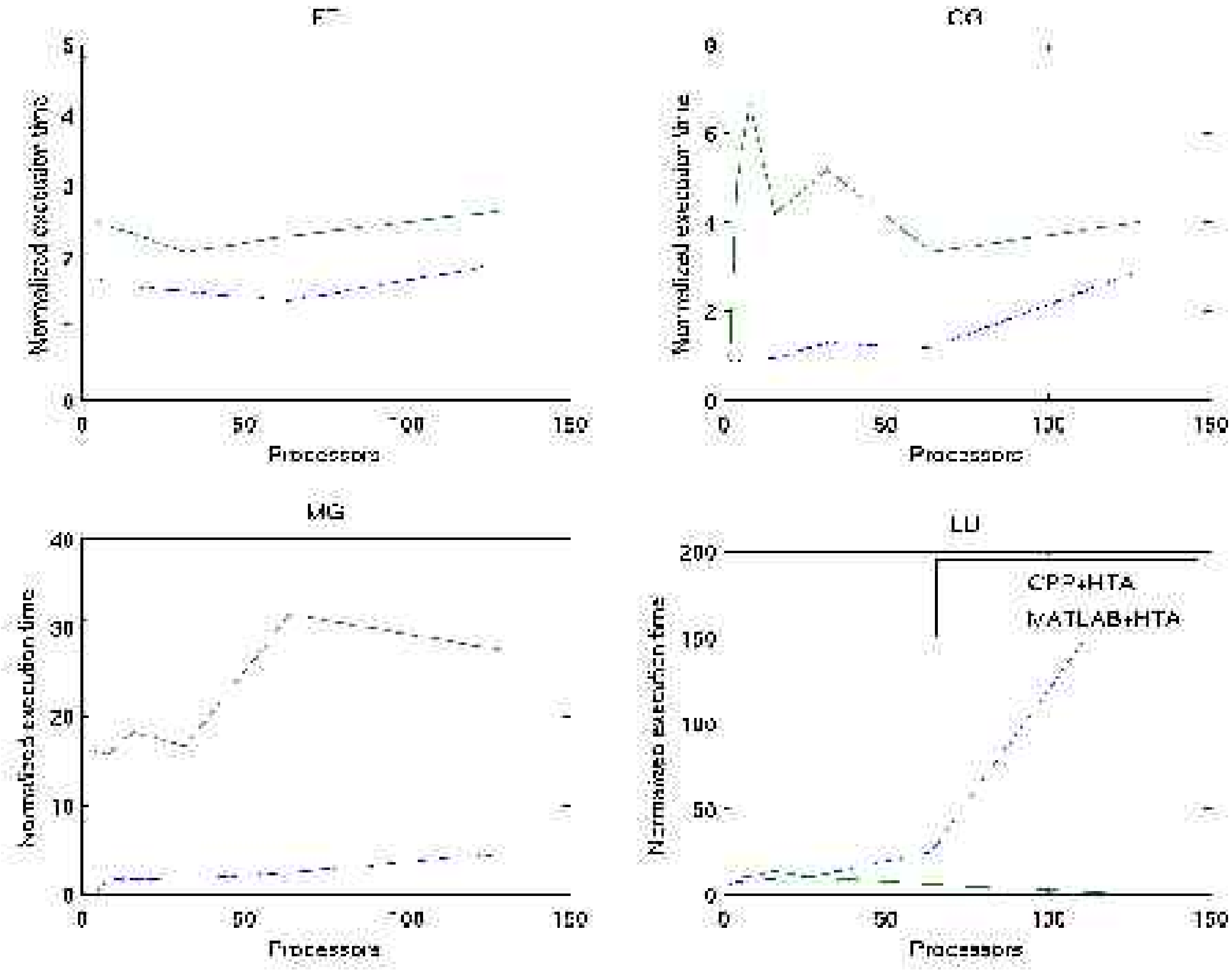
Outline

- Motivation
- HTA & Prior results
- Design of htalib & new features
- Experimental results ←
- Conclusion & Future work

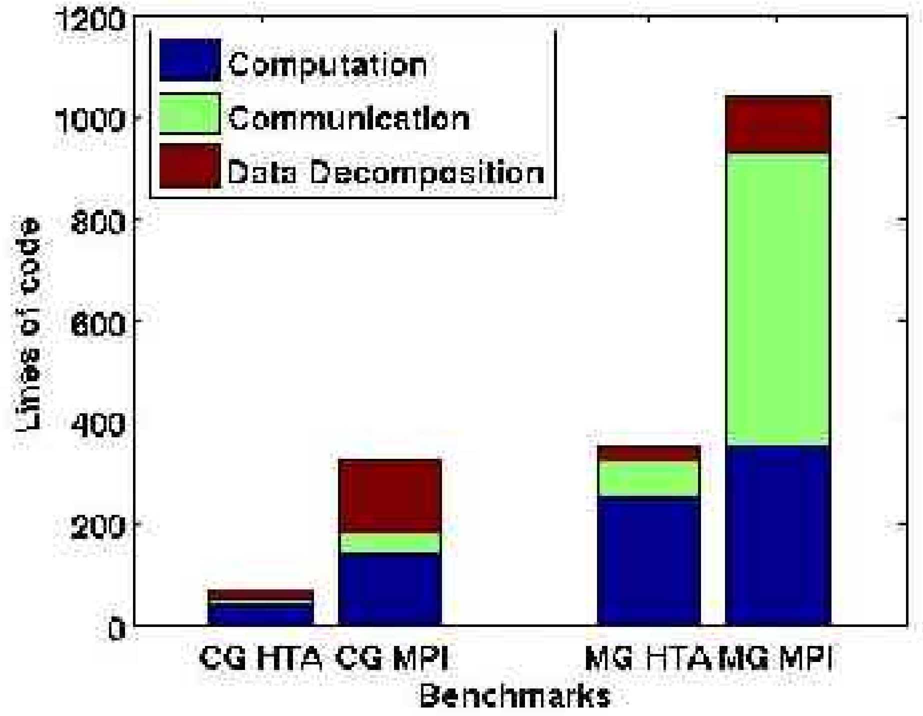
C++ results: NPB - CLASS B – BlueGene – Running time



C++ library vs MATLAB library



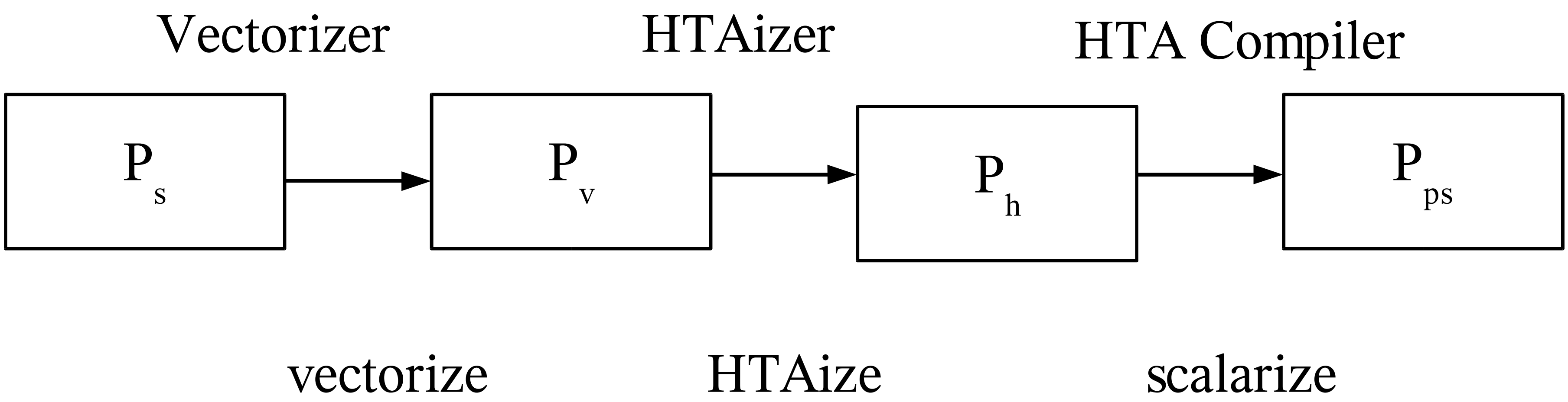
Lines of code measurement



Conclusions

- HTA treats tiles as first class language constructs
- HTA programs are efficient, readable and shorter
  - C++ run time library for HTA
  - new program constructs

Future Work


Vectorizer
HTAizer
HTA Compiler
P s
P v
P h
P ps
vectorize
HTAize
scalarize

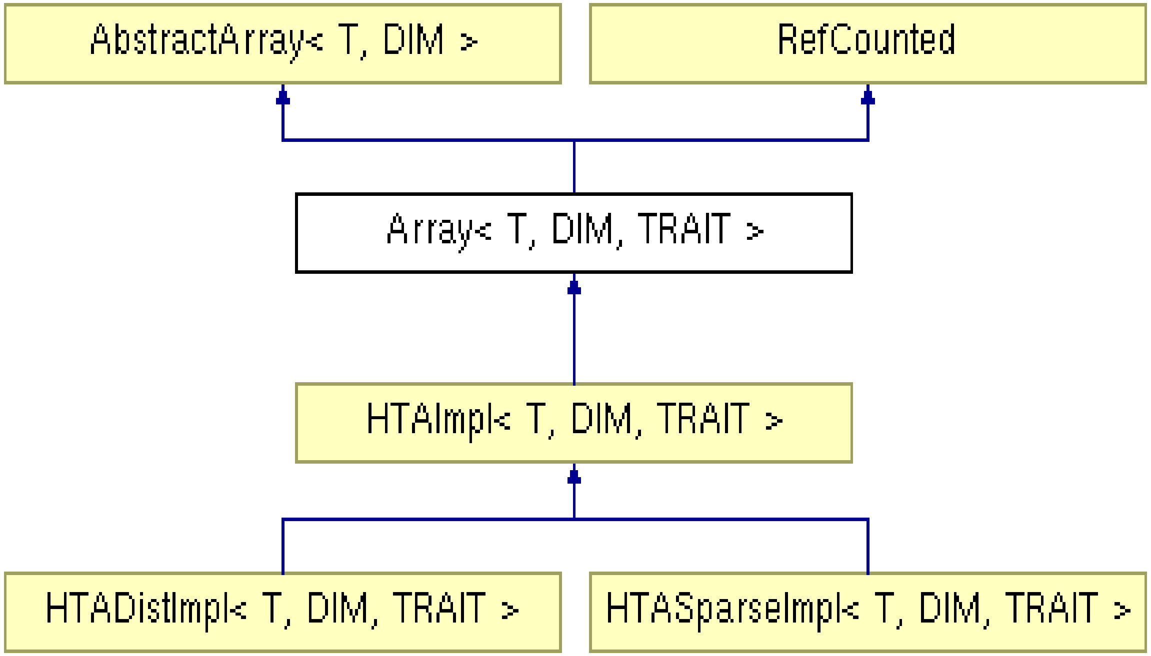
Acknowledgments

- Gheorge Almasi (IBM T. J. Watson Research Center)
- Calin Cascaval (IBM T. J. Watson Research Center)
- Nancy Amato (Texas A&M University)

data layering (from gabriel)

- Slide 1:
  - motivation, AMR example and MG benchmark example ; point out that the grids used by MG are a hierarchy of arrays or a “hierarchical array”
  - Say that currently users declare arrays of arrays(arrays of HTAs in our case; when going to AMR managing this array of arrays is geting complicated;
  - Slide 2 : one solution is data layering ; where each level of the HTA contains data and metadata about tiling at the level below;
  - Slide 3: mention that the multilayered HTA will assist the user
    - in maintaining the hierarchy of arrays
    - Compute the index spaces at coarse/refined grids to perform initialization of the data at refined level based on data at the coarse level and the other way;
    - Providing primitives to take out one level of the multilayered HTA and use it as a regular HTA to reuse algorithms currently available;

C++ class hierarchy



HTA for stencil computation

- Iterative PDE solvers
  - Computations on neighboring points
  - Benefit from tiling
- Current HTA code
  - Extra statements to update the shadow regions

```
B(1:n)[0]=B(0:n-1)[d];
B(0:n-1)[d+1]=B(1:n)[1];
A()[1:d]=S*(B()[2:d+1]+B()[0:d-1]);
```

Boundary exchanges

Code example

```
A=HTA<double,1>alloc(tiling,array,
ROW, ol);
```

```
A()[All]=S*(B()[All+1]+B()[All-1]);
```

- Shadow region consistency
  - Handled by htalib
  - Owner tile uses update-on-write policy
- Advantages
  - No extra boundary exchanges
  - Clean indexing syntax

Motivation

- Parallel Programming is a 2-stage process
  - a) write an optimal serial program
  - b) write an optimal parallel program
- Transition from a to b is a non-trivial task
  - a variety of data distribution
  - a variety of work distribution
  - a variety of communication & synchronization

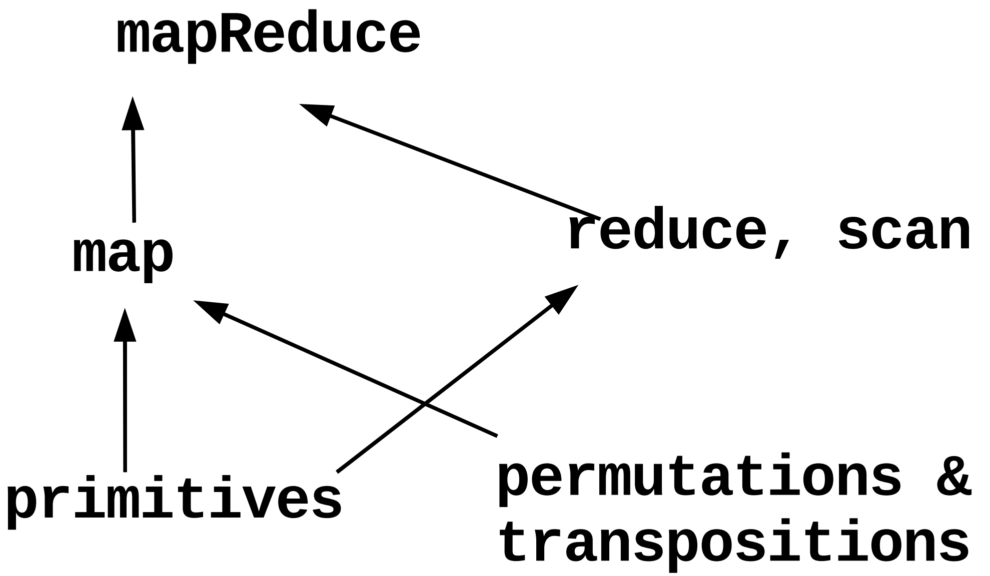
a variety of parallelization strategies

Research Efforts (II) : Languages & Libraries

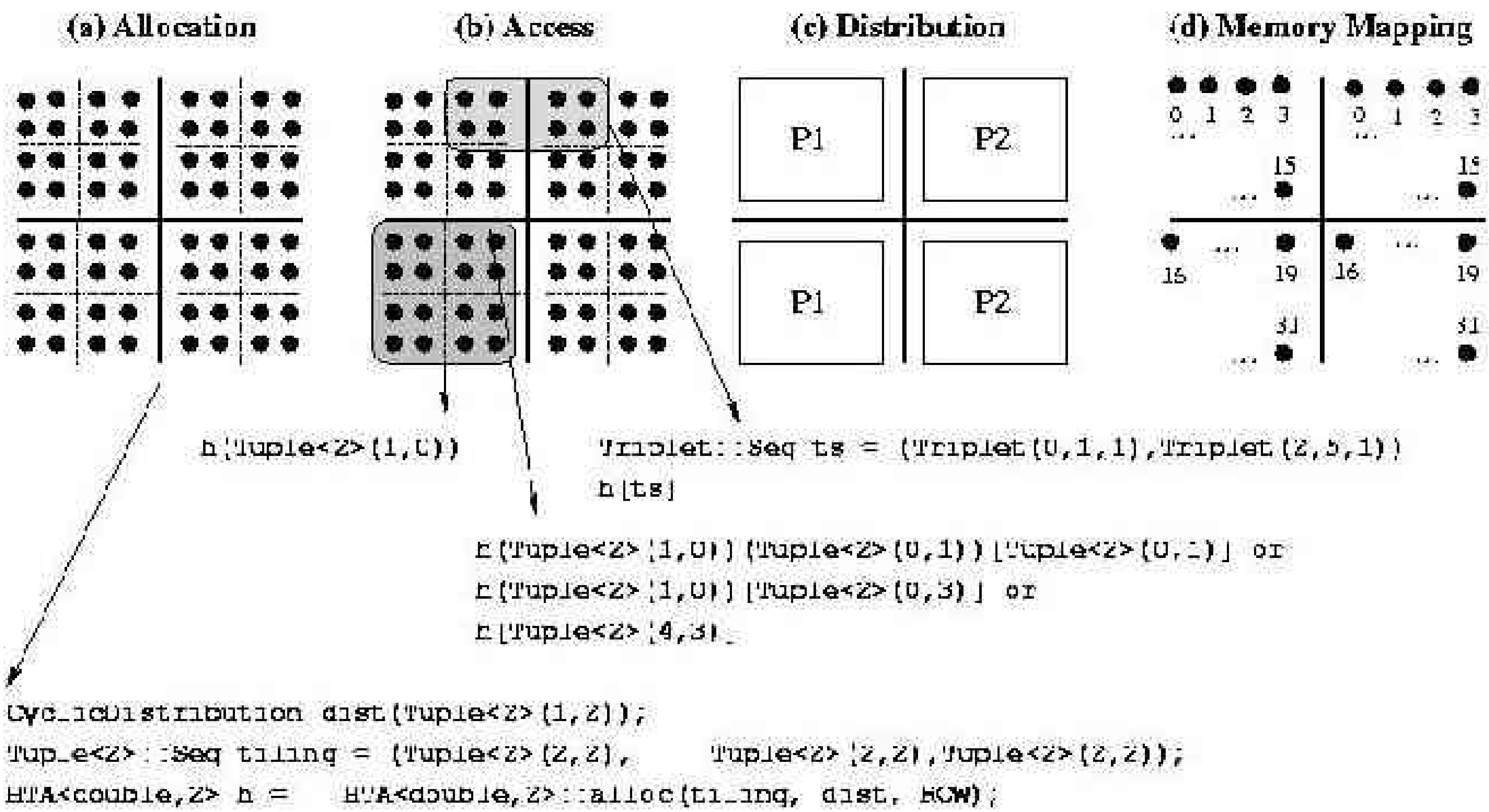
- Offer global view and/or single threaded model
  - ZPL – both
  - CAF, X10, UPC – PGAS + multi threaded model
  - POOMA (library) - both
- Syntactically identifiable communication

  None has been widely accepted

Operator Framework



Other classes



Map-Reduce (cont.)

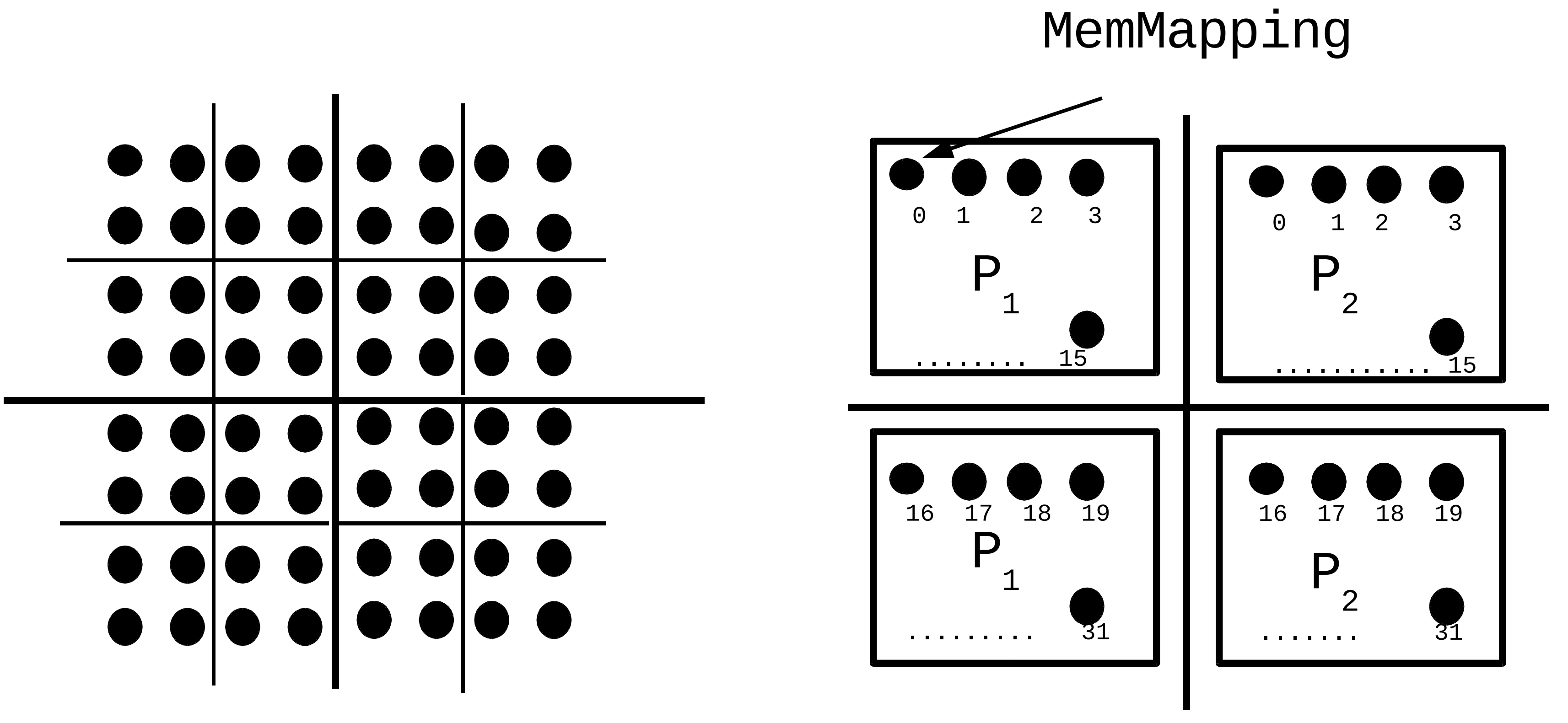
```
template<typename T, int N>
struct MaxN : public Operator::MapReduce<T, XchgData<T, N>>
{
  void reduce (const XchgData<T, N>& d) {
    for (int i = 0; i < N; i++)
      map(d.maxKey[i], d.maxVal[i]);
  }

  void map (const Tuple& key, const T& val) {
     //code to accumlate the minimum val and its key
  }

  const XchgData<T, N>& result () const { return buf_; }
  XchgData<T, N> buf;
};
```

```
        MaxN<double, 2> mr = MaxN <double, 2>();
        hta.mapReduce(mr);
```

HTA Internal Representation




```
CyclicDistribution<2> dist (Tuple<2>(1,2));
Tuple<2>::Seq tiling = (Tuple<2>(2,2), Tuple<2>(2,2), Tuple<2>(2,2));
HTA<double, 2> h = HTA<double,2>(tiling, dist, ROW);
```

Advantages

- Easy to write a wide range of performance-conscious programs
  - Shared and Distributed memory programs (MIMD)
  - Vector programs (SIMD)
  - Cache-conscious programs

Advantages

- Minimal compiler involvement
  - No complex analysis to unleash hidden parallelism
  - Unit of operation being tiles, message vectorization comes for free
- Natural to represent multi-level (nested) parallelism