A Parallel Numerical Solver Using Hierarchically Tiled Arrays

James Brodman, G. Carl Evans, Murat Manguoglu, Ahmed Sameh, María J. Garzarán, and David Padua

UPCRC Illinois
Universal Parallel Computing Research Center

Outline

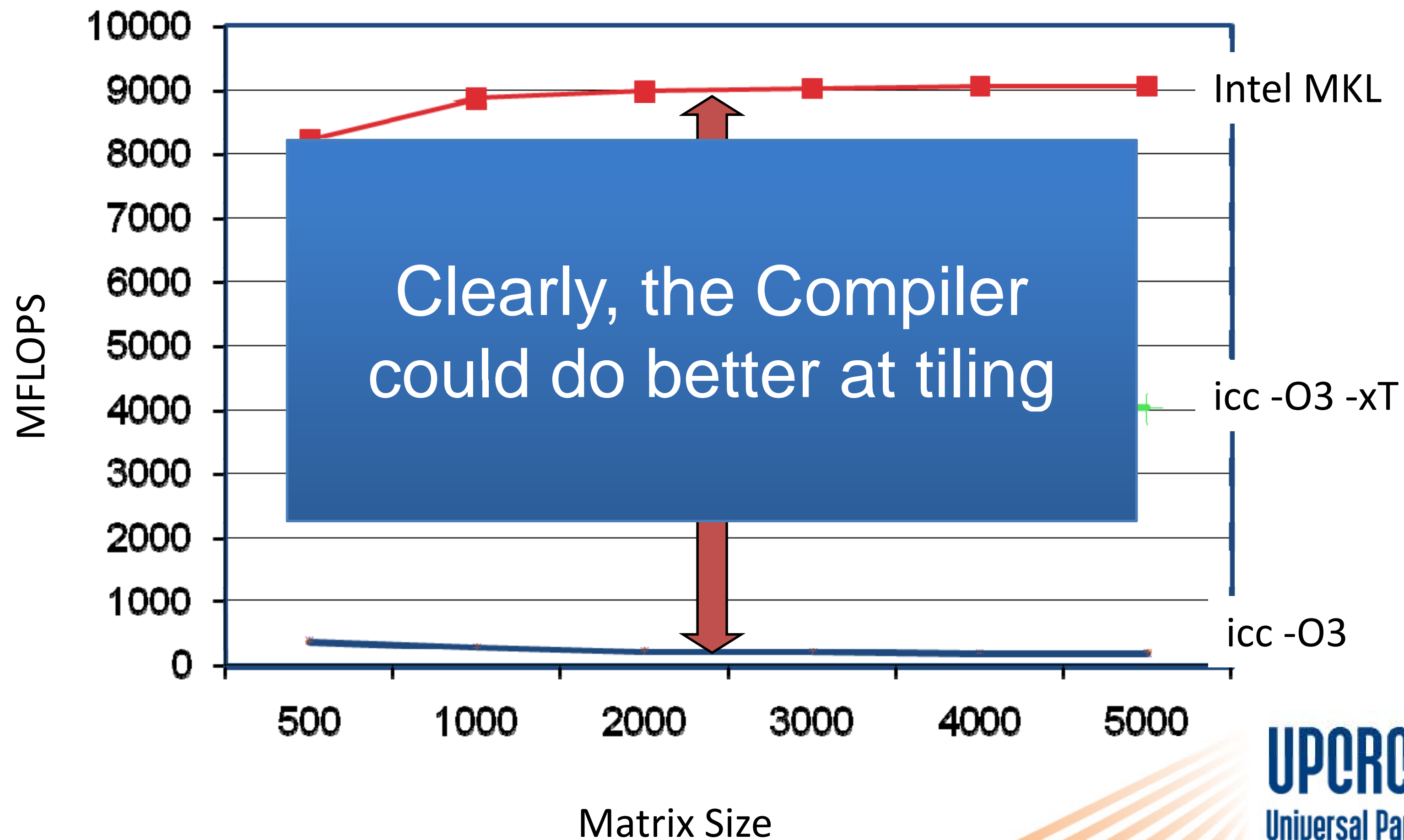
- Motivation
- Hierarchically Tiled Arrays (HTAs)
- SPIKE
- HTA-SPIKE
- Conclusions

UPCRC Illinois
Universal Parallel Computing Research Center

Motivation

- Still much room for advancement in parallel programming
- Want to facilitate parallel programming but still provide control over performance
  - Raise the level of abstraction
    - Higher level data parallel operators and their associated aggregate data types
    - Write programs as a sequence of operators
      - Can easily reason about the resulting program
      - Portable across any platform that implements the desired operators
  - Control performance through:
    - Data Parallelism
    - Hierarchical Tiling
      - Who should tile?

Tiling and Compilers (Matrix Multiplication)



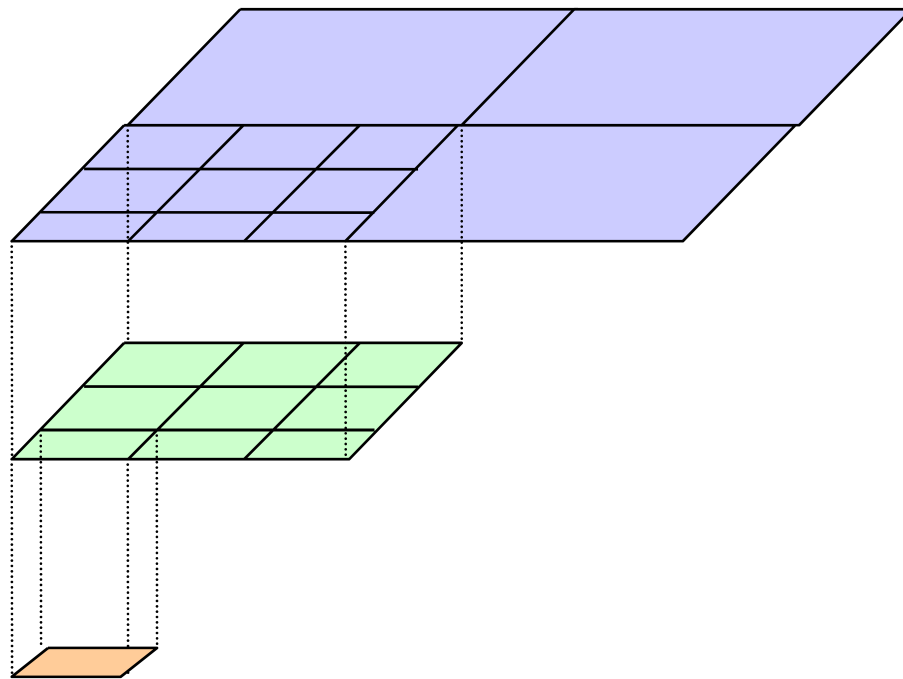
UPCRC Illinois
Universal Parallel Computing Research Center

Hierarchically Tiled Array

- The Hierarchically Tiled Array, or HTA, is a data type based on higher level data parallel operators
  - Fortran 90 array operators
  - Hierarchical Tiling
- Makes Tiles first class objects
  - Recognizes importance of tiling to control:
    - Locality
    - Data Distribution
  - Referenced explicitly, not implicitly generated by the compiler
  - Operators extended to function on tiles
- C++ Library Implementations:
  - Distributed-Memory using MPI
  - Shared-Memory using TBB

UPCRC Illinois
Universal Parallel Computing Research Center

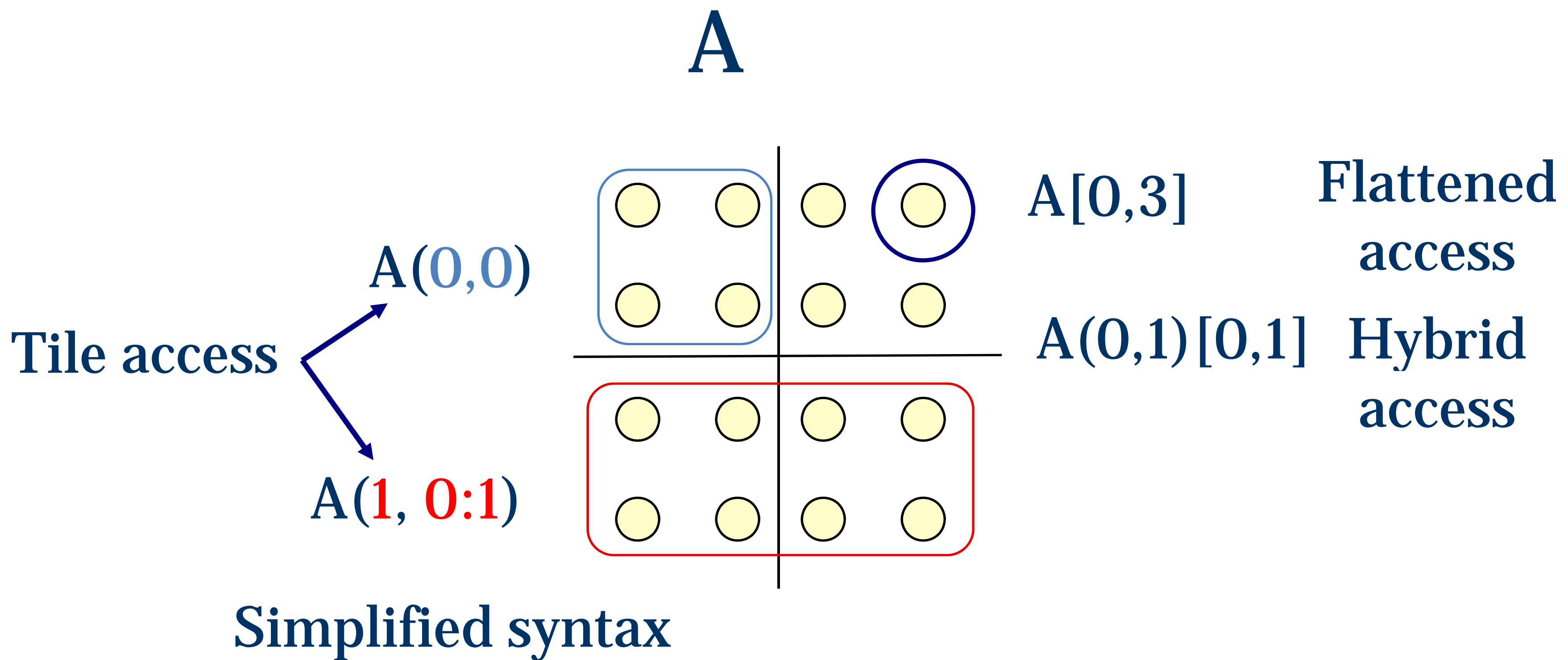
HTAs, Illustrated



Cluster	Memory Hierarchy
Cluster Node	L2
Multicore	L1
Cache	Register

UPCRC Illinois
Universal Parallel Computing Research Center

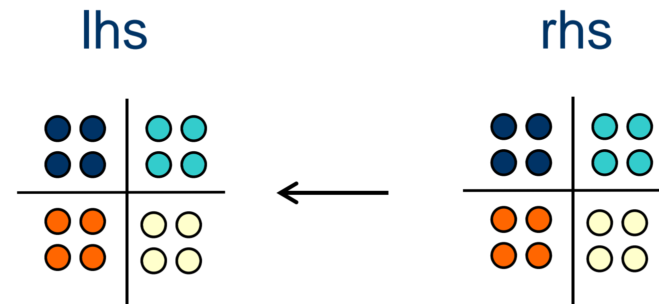
HTA Indexing



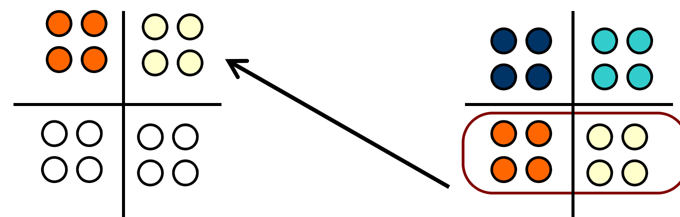
UPCRC Illinois
Universal Parallel Computing Research Center

HTA Assignment

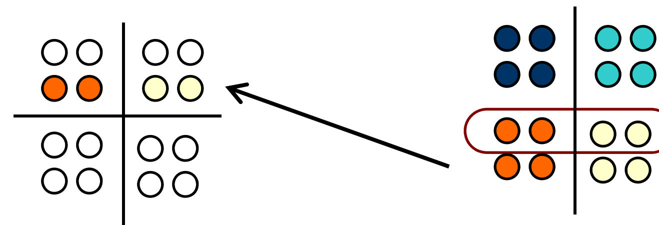
```
lhs      = rhs;
lhs(:,:) = rhs(:,:);
```




```
lhs(0,:) = rhs(1,:);
```

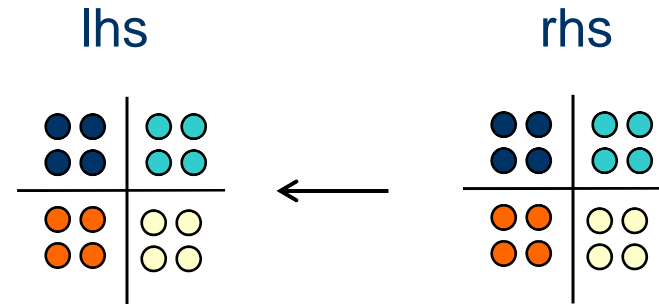


```
lhs(0,:)[1,:] = rhs(1,:)[0,:];
```



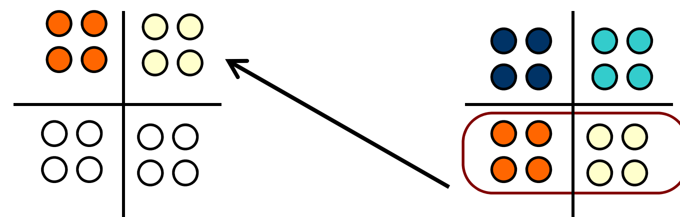
UPCRC Illinois
Universal Parallel Computing Research Center

HTA Assignment

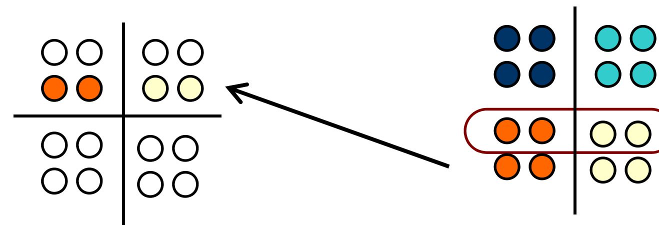


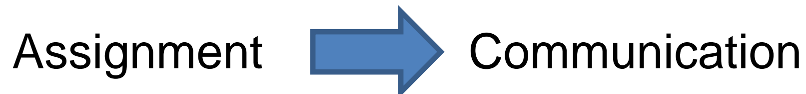

```
lhs      = rhs;
lhs(:,:) = rhs(:,:);
```

```
lhs(0,:) = rhs(1,:);
```



```
lhs(0,:)[1,:] = rhs(1,:)[0,:];
```





UPCRC Illinois
Universal Parallel Computing
Research Center

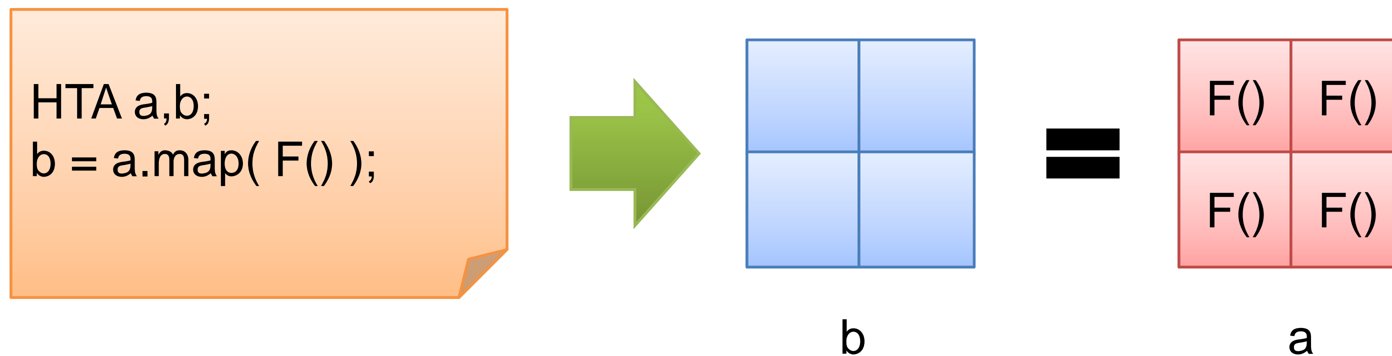
Higher Level HTA Operations

- Element-by-element operations
  - +,-,*,/,...
- Reductions
- Transpositions
  - Tile
  - Data
- MapReduce
- Matrix Multiplication
- FFT
- etc.

UPCRC Illinois
Universal Parallel Computing Research Center

User-Defined Operations

- Programmers can create new complex parallel operators through the primitive hmap
  - Applies user defined operators to each tile of the HTA
    - And corresponding tiles if multiple HTAs are involved
  - Operator applied in parallel across tiles
  - Assumes operation on tiles is independent



UPCRC Illinois
Universal Parallel Computing Research Center

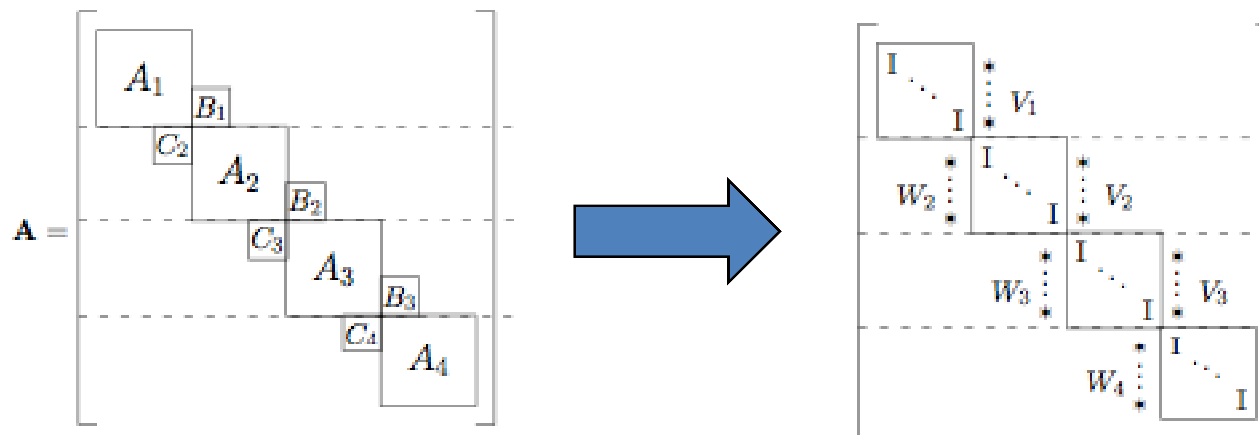
SPIKE

- Parallel solver for banded linear systems of equations
- SPIKE family of algorithms originally by Prof. Ahmed Sameh
- http://software.intel.com/en-us/articles/intel-adaptive-spike-based-solver/
- SPIKE's blocks map naturally to tiles in HTA

UPCRC Illinois
Universal Parallel Computing Research Center

SPIKE Algorithm

- Name derived from the matrix formed by multiplying the original block tri-diagonal system by the inverse of the blocks of A



UPCRC Illinois
Universal Parallel Computing
Research Center

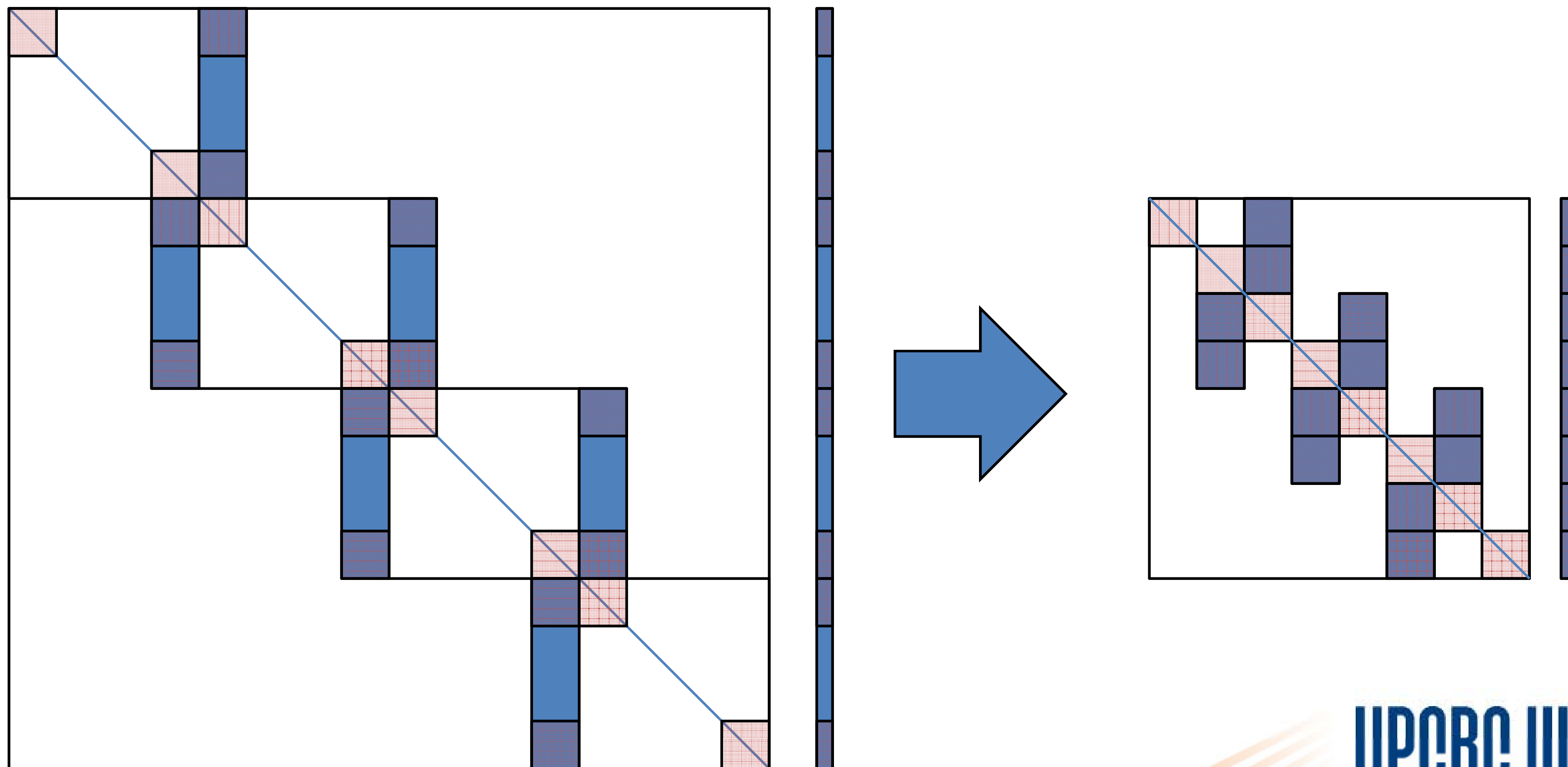
SPIKE Algorithm 2

- Most SPIKE algorithms proceed as follows:
  1. Compute the Spike matrix, S, in parallel
     - Compute LU factorization and solve instead of directly computing inverse
     - LAPACK routines DGBTRF/DGBTRS
  2. Form a reduced system of much smaller size
  3. Solve the reduced system directly
  4. Retrieve the global solution in parallel

UPCRC Illinois
Universal Parallel Computing Research Center

SPIKE Reduced System

- Extracts smaller, independent subsystem from the original system

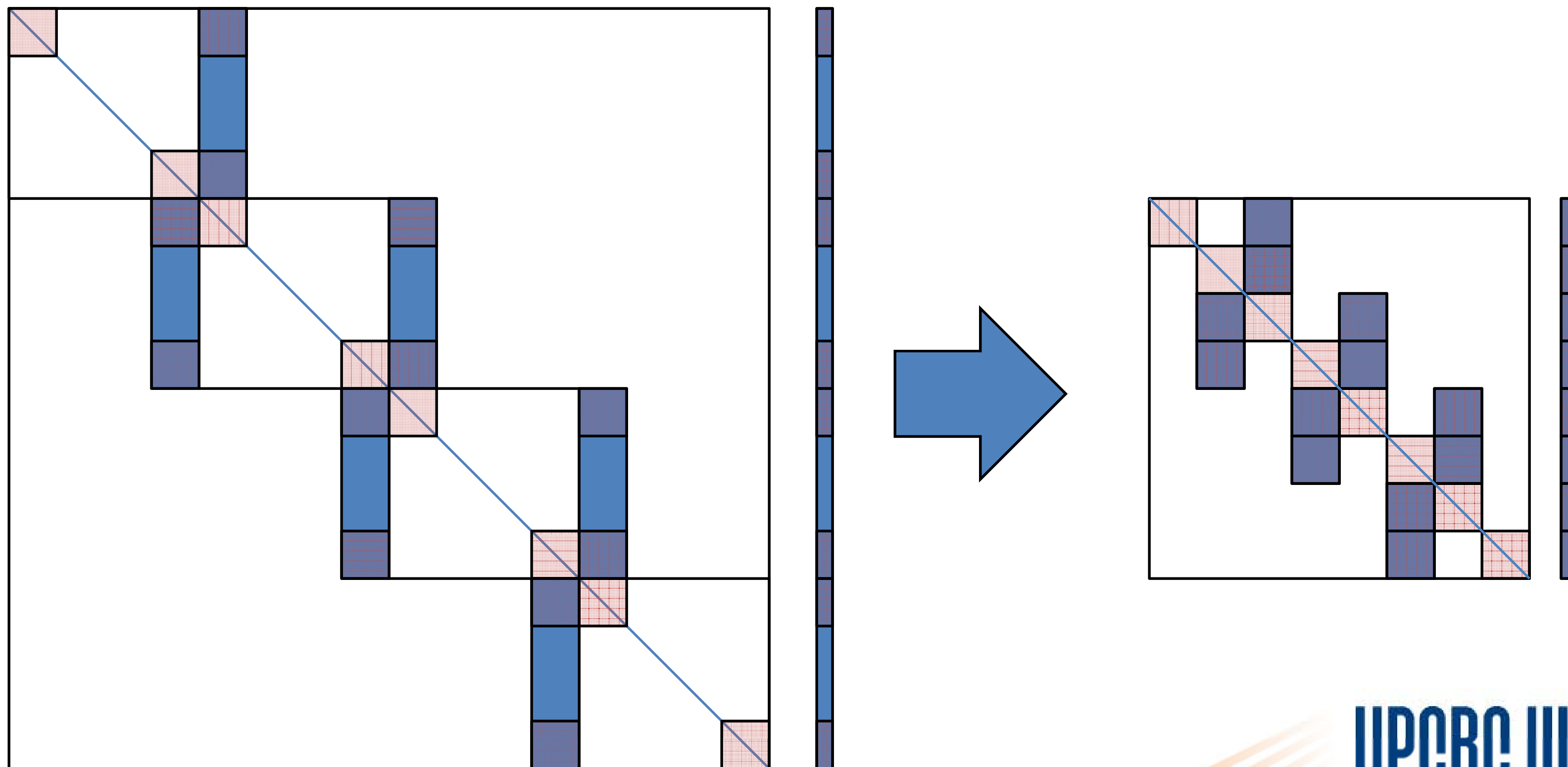


UPCRC Illinois
Universal Parallel Computing Research Center

SPIKE TU and TA

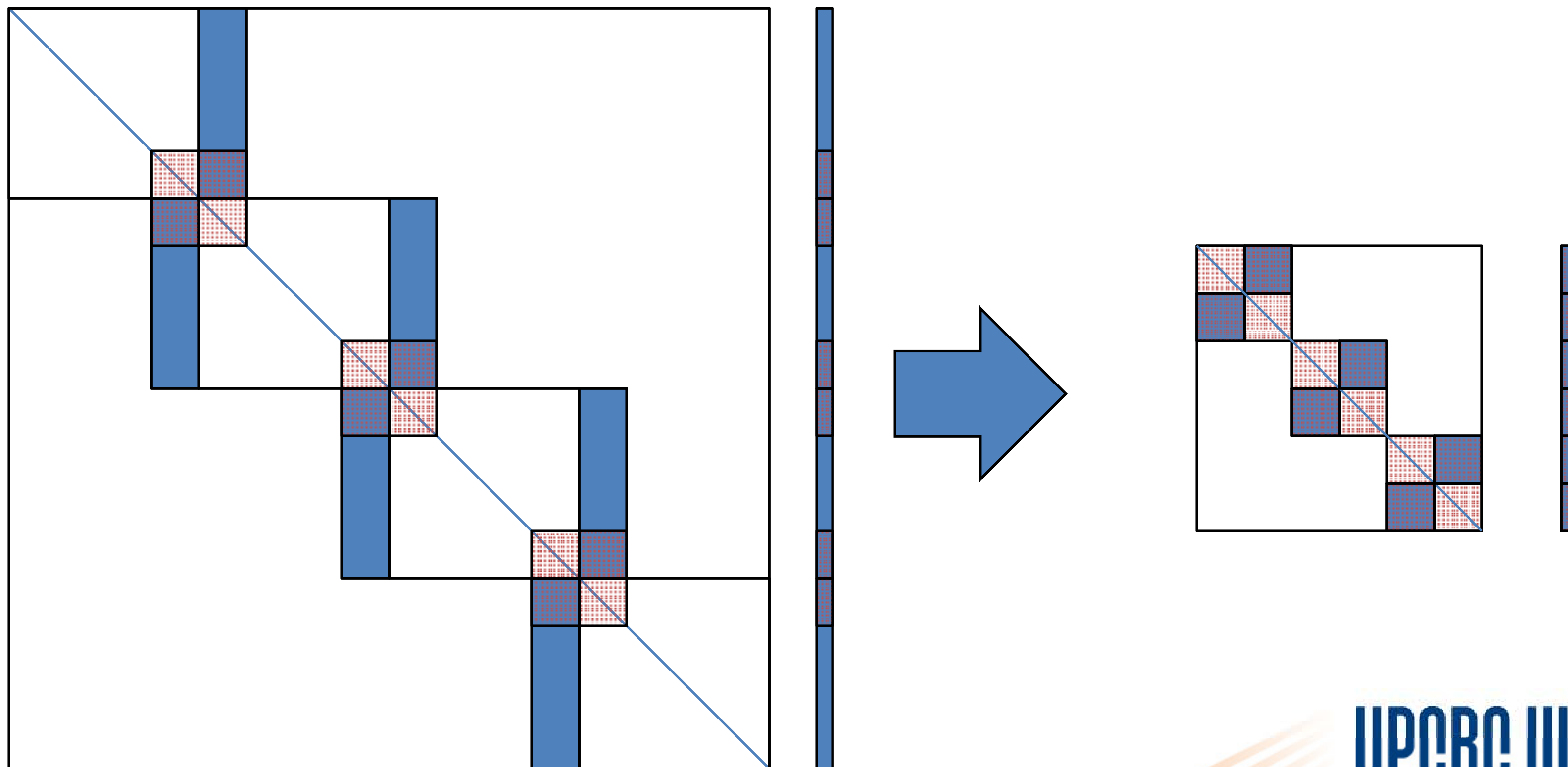
- SPIKE has several algorithmic variants
- Recent focus has been on the variants referred to as "Truncated"
- Faster than the full SPIKE algorithm
  - Reduces computation and has greater parallelism
  - Only applicable to diagonally dominant systems
    - Can wrap with an iterative method for non-diagonally dominant systems

Truncated Savings 1



UPCRC Illinois
Universal Parallel Computing
Research Center

Truncated Savings 2



UPCRC Illinois
Universal Parallel Computing
Research Center

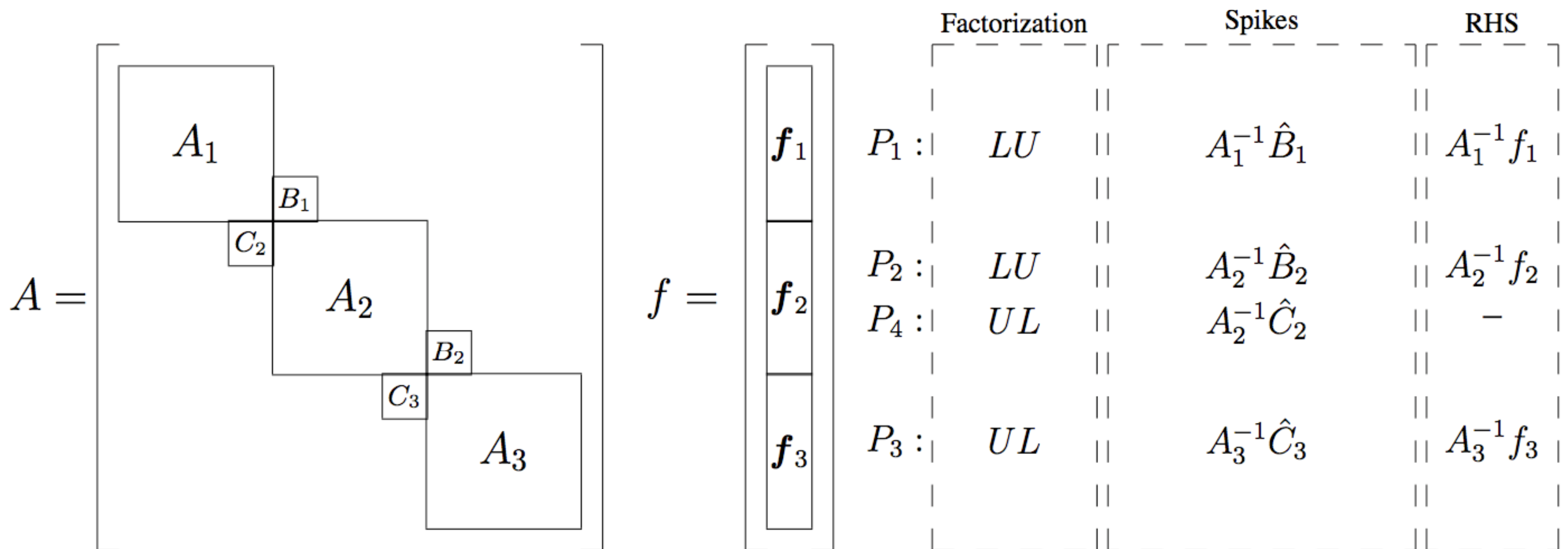
TU and TA

The truncated variants come in several flavors the the two that we implemented are TU and TA

- T – truncated
- U – Uses LU *and* UL factorizations
- A – Uses alternating LU *and* UL factorizations

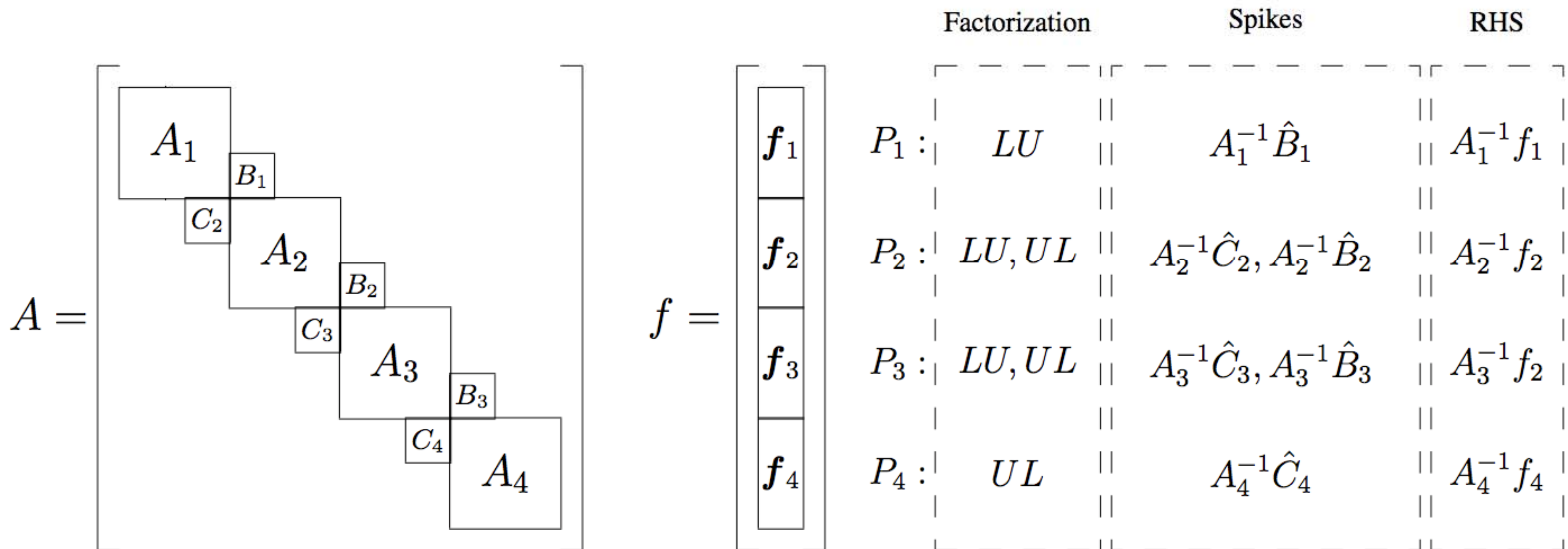
The difference is in how the work is distributed to the processors not in what work is done.

TA Work Distribution



UPCRC Illinois
Universal Parallel Computing Research Center

TU Work Distribution

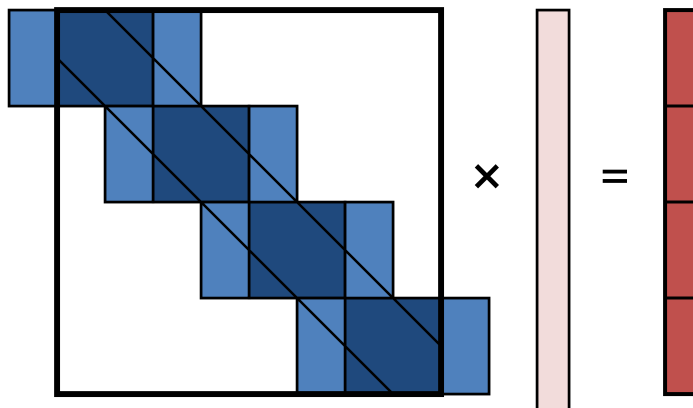


UPCRC Illinois
Universal Parallel Computing
Research Center

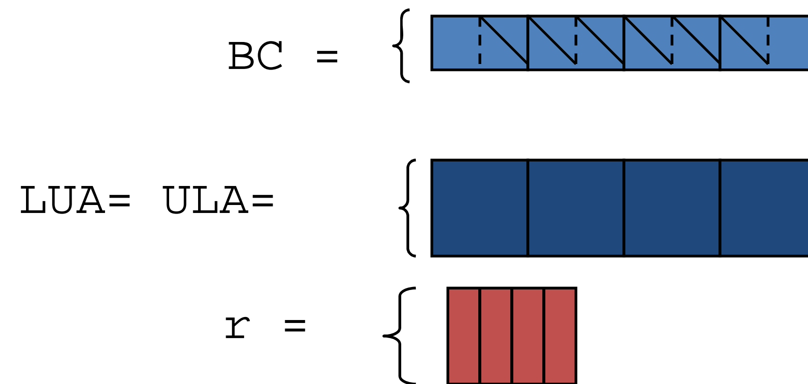
HTA SPIKE

- Blocks of SPIKE map very well to HTA tiles
- Factorization and Solving of tiles parallelized with hmap
- Reduced System formed using array assignments

Original Matrix System



HTA Representation



UPCRC Illinois
Universal Parallel Computing Research Center

TU Implementation

```
...

LUA.hmap(factorize_lua());
ULA.hmap(factorize_ula());                                // factorize blocks of A

BC.hmap(solve_bc(),LUA,ULA);                              // calculate the spike tips W(t) and V(b) from Bs and Cs

g.hmap(solve_lua(),LUA);                                  // update right hand side

REDUCED()[0:m-1,m:2*m-1] =  BC(0:blks-2)[0:m-1,0:m-1];    // form the reduced system
REDUCED()[m:2*m-1,0:m-1] =  BC(1:blks-1)[0:m-1,m:2*m-1];

greduced()[0:m-1] = g(0:blks-2)[blocksize-m:blocksize-1]; // form the reduced system RHS
greduced()[m:2*m-1] = g(1:blks-1)[0:m-1];

REDUCED.hmap(factorize());                                // factorize the reduced system

greduced.hmap(solve(),REDUCED);                           // solve the reduced system

fv = r(0:num_blocks-2); fr_half = greduced()[0:m-1];      // Update RHS as r = r - Bz - Cz
B.map(dgemv(),fv,fr_half);
r(0:num_blocks-2) = fv;
fw = r(1:num_blocks-1); fr_half = greduced()[m:2*m-1];
C.map(dgemv(),fw, fr_half);
r(1:num_blocks-1) = fw;

r.hmap(solve_lua(),LUA);                                  // Solve the updated system
...
```

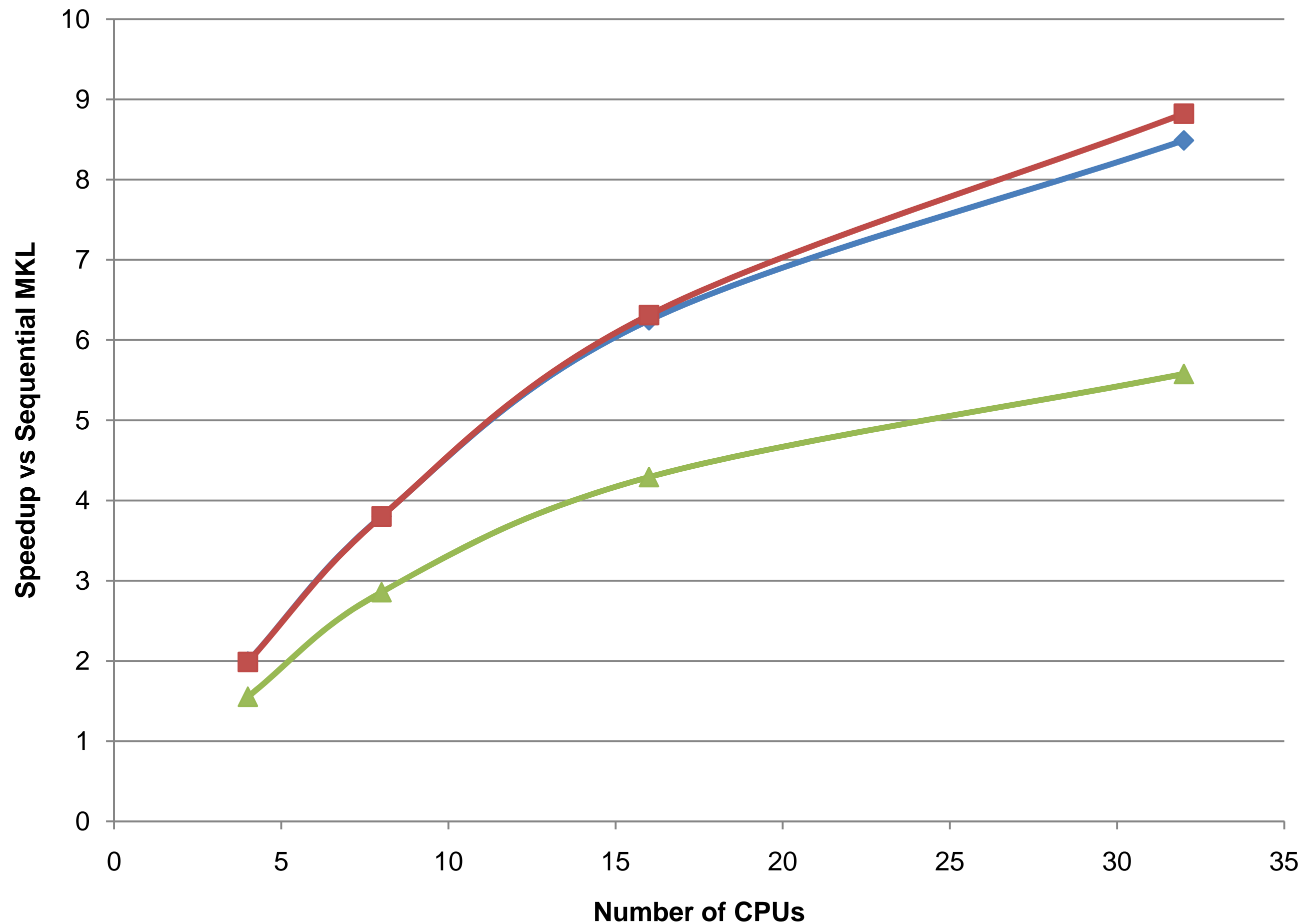

UPCRC Illinois
Universal Parallel Computing
Research Center

Portability

- The HTA programming model expresses programs as a sequence of higher level data parallel operators
  - Portable as long as the appropriate implementations of the operators exist for the target platform
  - For SPIKE, array assignments, tile indexing, and hmap.
- The code on the last slide runs on both the MPI and TBB implementations of the HTA library
  - Simply change which header to include.

UPCRC Illinois
Universal Parallel Computing
Research Center

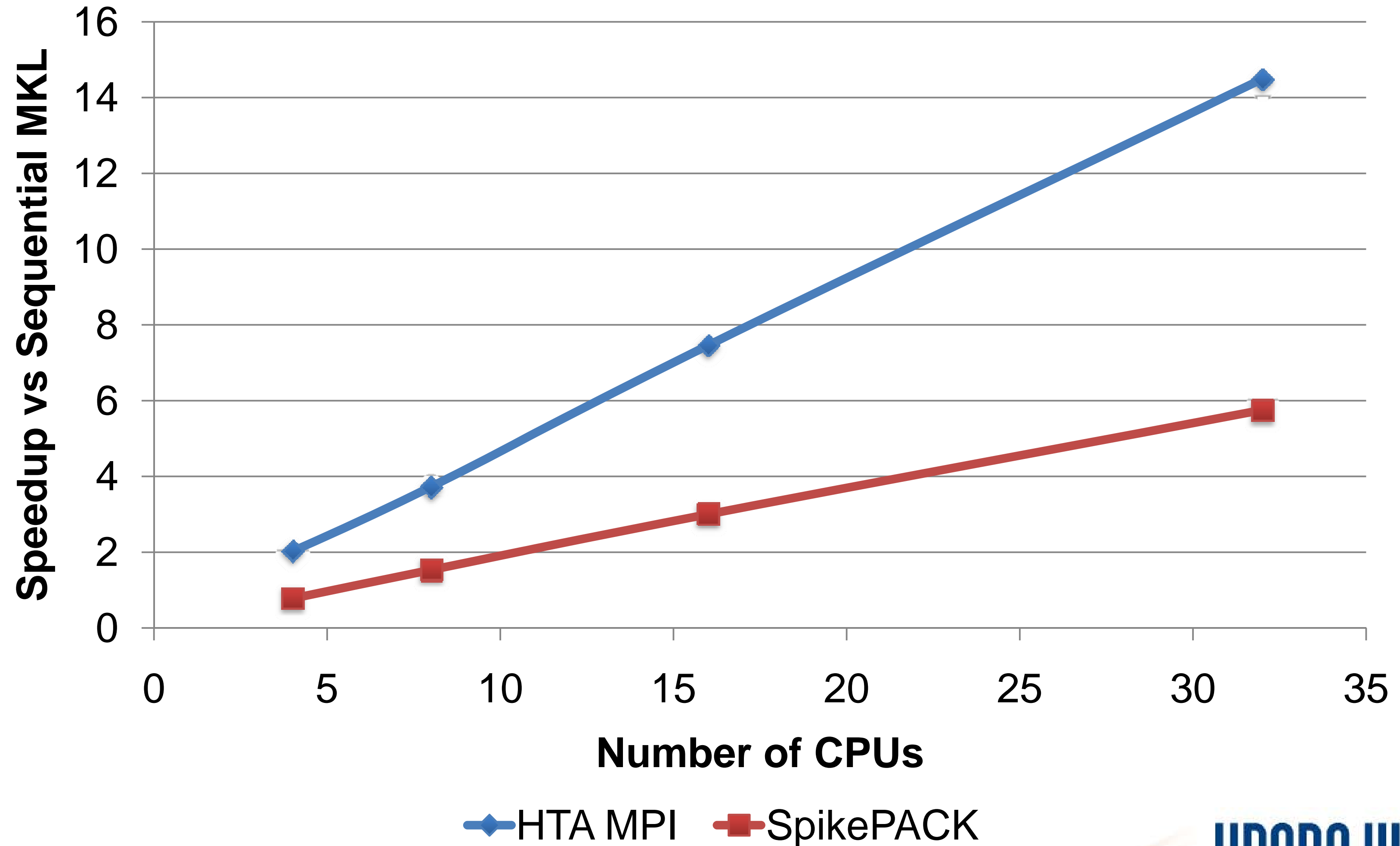
SPIKE TU Performance



32 Core Intel Xeon 7555 (Nehalem) @ 1.86 GHz

UPCRC Illinois
Universal Parallel Computing Research Center

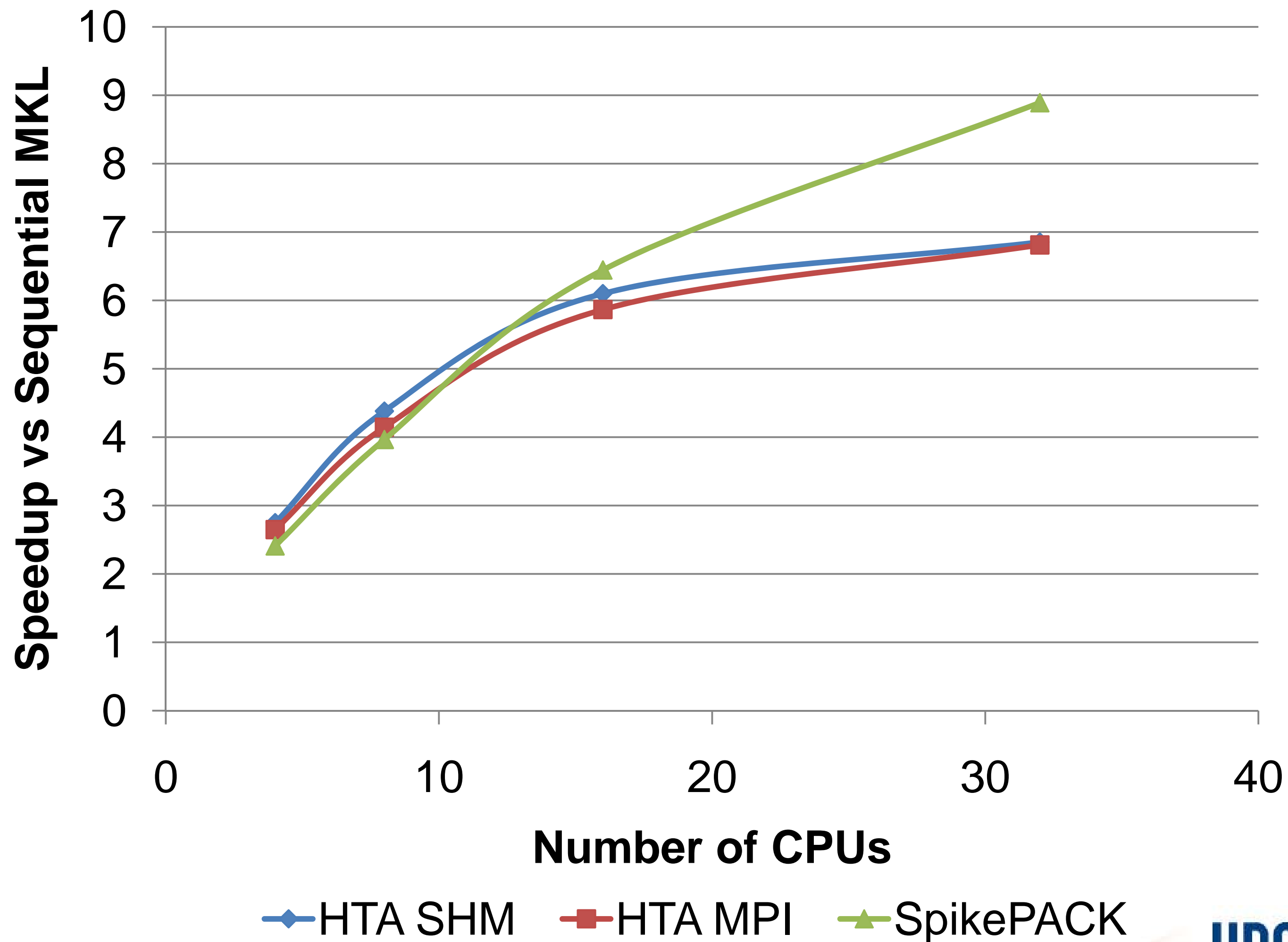
Cluster TU Performance



Intel Xeon X5550 (Nehalem) @ 2.66 GHz

UPCRC Illinois
Universal Parallel Computing Research Center

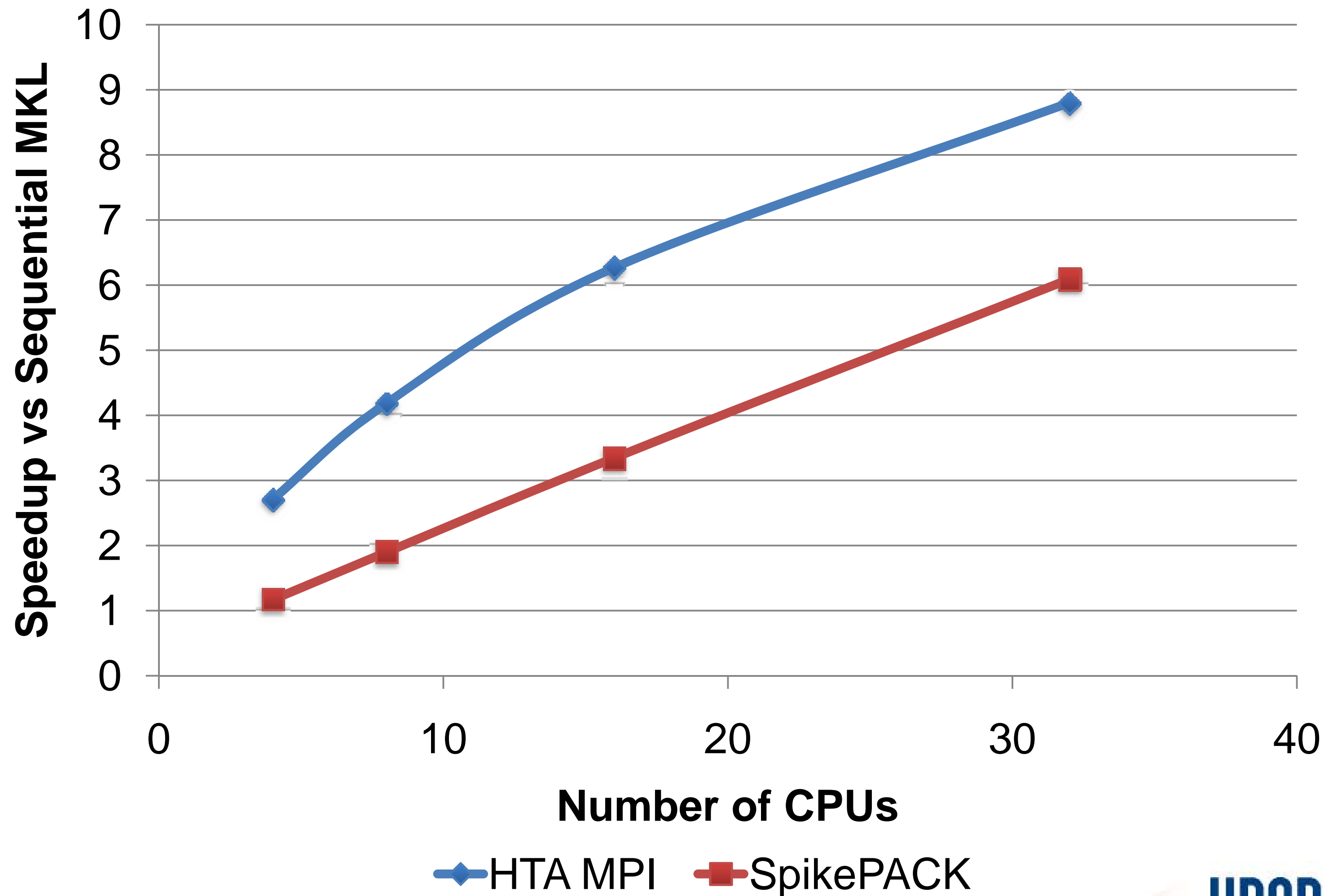
SPIKE TA Performance



32 Core Intel Xeon 7555 (Nehalem) @ 1.86 GHz

UPCRC Illinois
Universal Parallel Computing Research Center

Cluster TA Performance



Intel Xeon X5550 (Nehalem) @ 2.66 GHz

UPCRC Illinois
Universal Parallel Computing Research Center

Conclusions

- The SPIKE algorithms are a good fit for the HTA programming model
  - First class tiles
  - Data parallel operators
  - Portable
- Clean, compact code
  - Our implementation takes ~50 lines of code
  - SpikePACK's Fortran+MPI? …..a lot more.
  - A challenge – one of the most complex algorithms we've looked at using HTAs
- Good performance
  - Original goal was only to match performance of Fortran+MPI

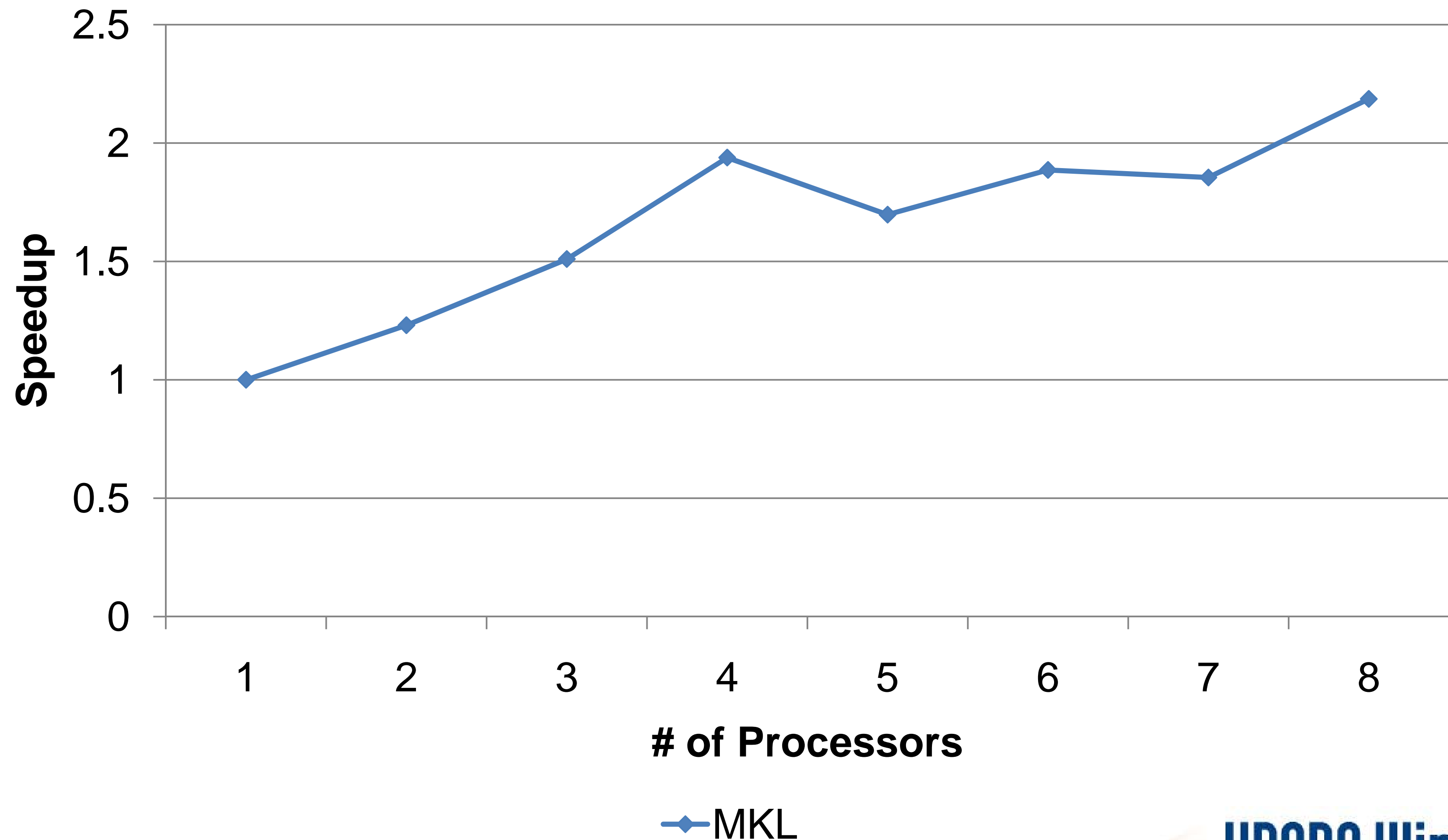
Future Work

- Sparse Tiling
  - Allow programmers to express the structure of the original system
- Support for Heterogeneous Architectures
  - Clusters of Multicores, GPUs
- Look at other complex algorithms
- Linear algebra libraries and data layout

Questions?

UPCRC Illinois
Universal Parallel Computing Research Center

Multithreaded MKL Performance



UPCRC Illinois
Universal Parallel Computing Research Center

Why is Raising the Level of Abstraction Good?

1) Allows programmers to focus on the algorithm rather than on the implementation

2) Data parallel programs resemble conventional, sequential programs
- – Parallelism is encapsulated
- – Parallelism is structured

3) Compact and readable

UPCRC Illinois
Universal Parallel Computing Research Center

Why is Raising the Level of Abstraction Good?

4) Portable

– Run on any class of machine with suitable implementations of the operators

– Multicore/Clusters/GPUs

5) Programmers have control of determinacy.

– Implicit barriers exist between operators. If operators do not alter any shared state, the program is deterministic.

6) Facilitates Optimization

– Exposes tunable parameters

UPCRC Illinois
Universal Parallel Computing Research Center