Data Parallel Programming in R

David Padua

Department of Computer Science
University of Illinois at Urbana-Champaign

Outline

- Parallelism
- Data Parallel Programming and abstractions
- Hierarchically Tiled Arrays
- Future plans

I. Parallelism

- Parallelism is crucial for
  - Continued gains in performance
  - Maximum performance at any given time
  - Also the most natural way to program for reactive computing (but not the topic of this presentation)
- Main problem with parallelism is productivity.
- Need the right languages, libraries and tools

II. Data parallel programming

- In its simplest form is just the execution of the same operation on each element of an aggregate (array, set, database relation).
- Sequential execution across these operations
- Crucial issue is what should these operations should look like (research problem)
- There are numerous proposals
  - Array operations (Iversion ca. 1960)
  - MapReduce (Google, ca. 2000)
  - Galois (Pingali, ca. 2000)

II. Data parallel programming Array Constructs

- Popular among scientists and engineers.
  - Fortran 90 and successors
  - MATLAB
  - R
- Parallelism not the reason for this notation.

II. Data parallel programming Array Constructs

- Convenient notation
  - Compact
  - Higher level of abstraction

```
do i=1,n
   do j=1,n
      C(i,j)= A(i,j)+B(i,j)
   end do
end do
```

```
C = A + B
```

```
do i=1,n
   do j=1,n
      S = S + A(i,j)
   end do
end do
```

```
S += sum(A)
```

II. Data parallel programming
Array constructs

- Used in the past for parallelism: Illiac IV, Connection machine
- Today: Intel’s Cilk (mainly for microprocessor vector extensions)

II. Data parallel programming Benefits - Programmability

- Da
  - – increasing the size of

Operations implemented as parallel loops in shared memory

- Data parallel programs using powerful operator resemble conventional, serial programs
  - Parall
  - Parall

Operations implemented as messages if distributed memory

- Portable
  - Can run on any class of machine for which the appropriate operators are impleme
    - Shared/Di

Operations implemented with vector intrinsics for SIMD

- Interoperates with R !

II. Data parallel programming Completeness

- Can all problems be solved in the **most efficient** manner with data parallel programming ?

- Most ? All ??
  - Other (lower level)forms must be there in the same way that we still use assembly language sometimes.

II. Data parallel programming Completeness

- Numerical computing
  - William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 1996. **Numerical Recipes in Fortran 90** (2nd Ed.): The Art of Parallel Scientific Computing. Cambridge University Press, New York, NY, USA.
- Graph algorithms
  - Aydın Buluç and John R Gilbert. 2011. **The Combinatorial BLAS: design, implementation, and applications**. Int. J. High Perform. Comput. Appl. 25, 4 (November 2011), 496-509.
  - Keshav Pingali, Donald Nguyen, Milind Kulkarni, Martin Burtscher, M. Amber Hassaan, Rashid Kaleem, Tsung-Hsien Lee, Andrew Lenharth, Roman Manevich, Mario Méndez-Lojo, Dimitrios Prountzos, and Xin Sui. 2011. **The tao of parallelism in algorithms**. In Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation (PLDI '11). ACM, New York, NY, USA, 12-25.
- Database algorithms
  - Anand Rajaraman and Jeff Ullman. **Mining of Massive Datasets** .Cambridge University Press, 2011.
- …

II. Data parallel programming Translating to SPMD

- SPMD is the notation of choice for distribute memory machines (and GPUs).
- Easy to convert from array notation to SPMD form.
- This is an optimization.

```
real a, b, x(1000)
a=sin (b)
x(:)=x(:)+a
```

$\Downarrow$

```
real a, b, x(1000/p)
/* a and b are replicated */
a=sin(b)
x(:)=x(:)+a
```

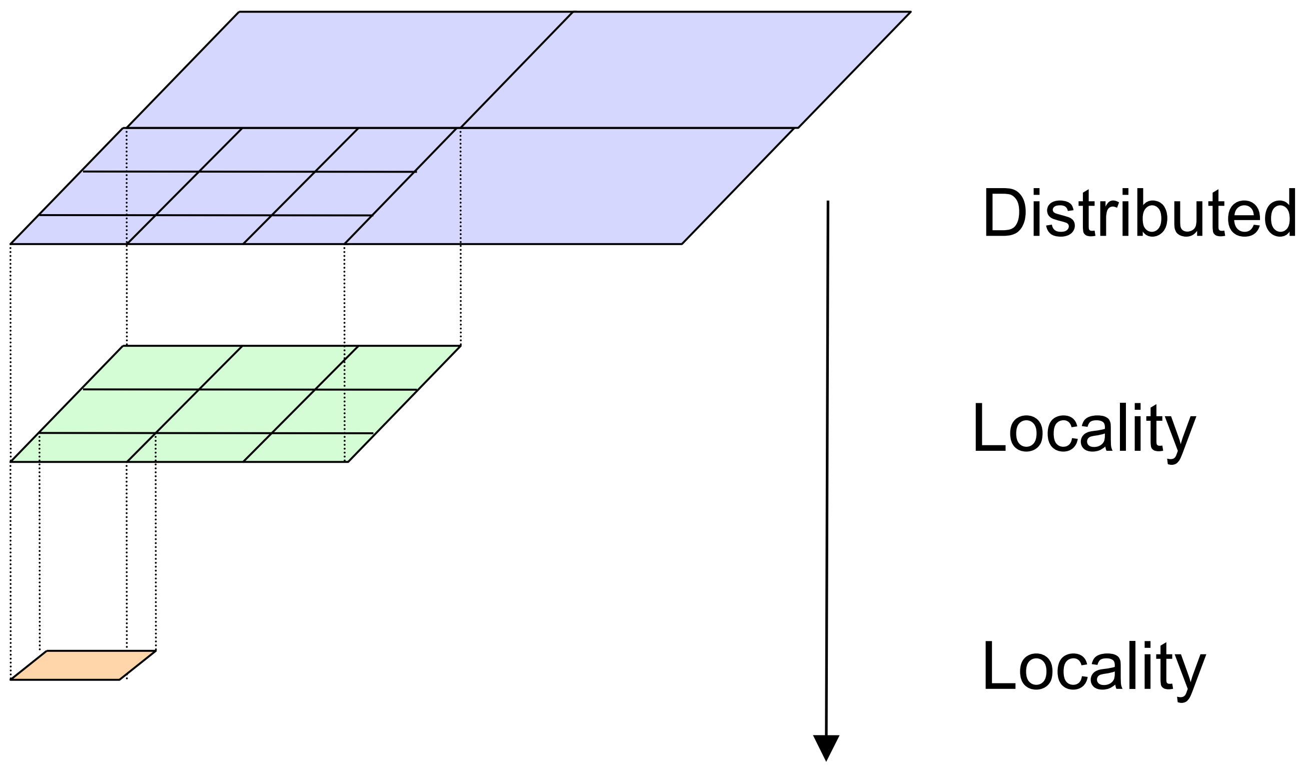
III. Hierarchically Tiled Arrays: Our data parallel notation for array computations

Hierarchically Tiled Arrays

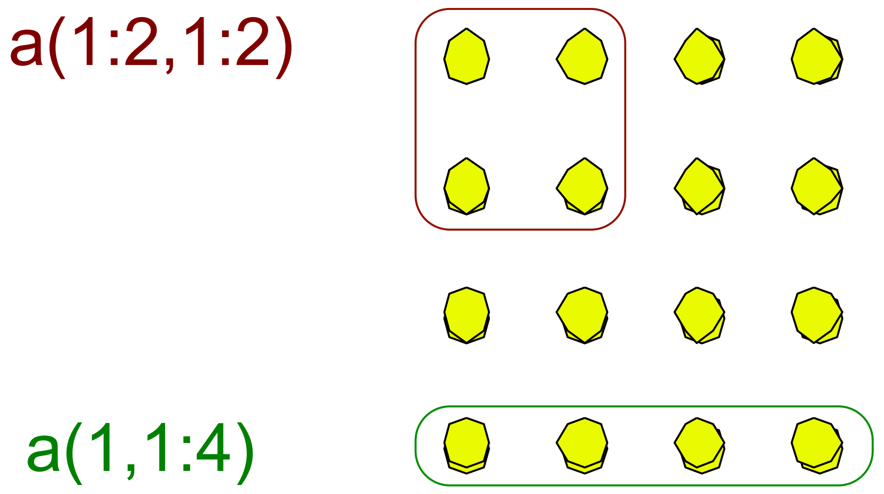
- Recognizes the importance of blocking/tiling for locality and parallel programming.
- Makes tiles first class objects.
  - Referenced explicitly.
  - Manipulated using array operations such as reductions, gather, etc..

G. Bikshandi, J. Guo, D. Hoeflinger, G. Almasi, B. Fraguela, M. Garzarán, D. Padua, and C. von Praun. Programming for Parallelism and Locality with Hierarchically Tiled. *PPoPP*, March 2006.

Hierarchically Tiled Arrays

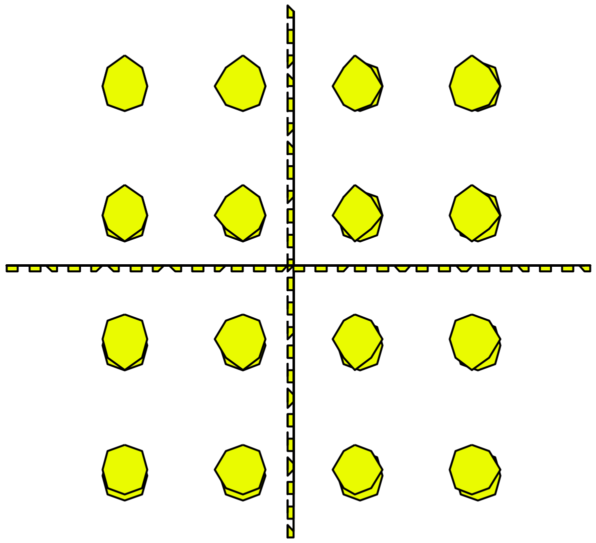


Vector Addressing



In general, $a(v_n)$

HTA Addressing

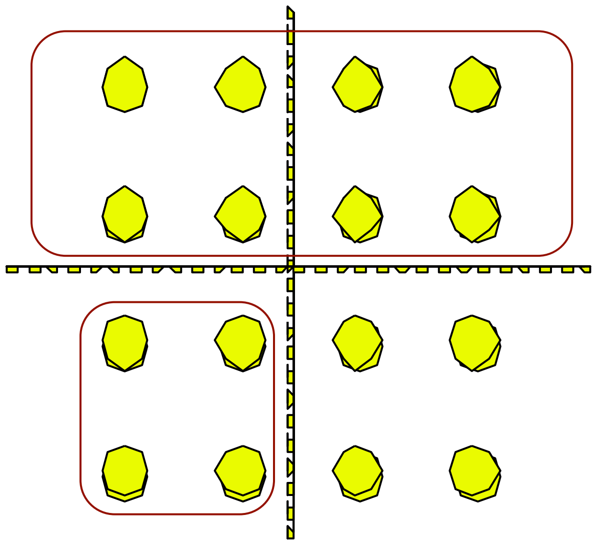


HTA Addressing

h{1,1:2} (hta)

h{2,1}   (array)

hierarchical

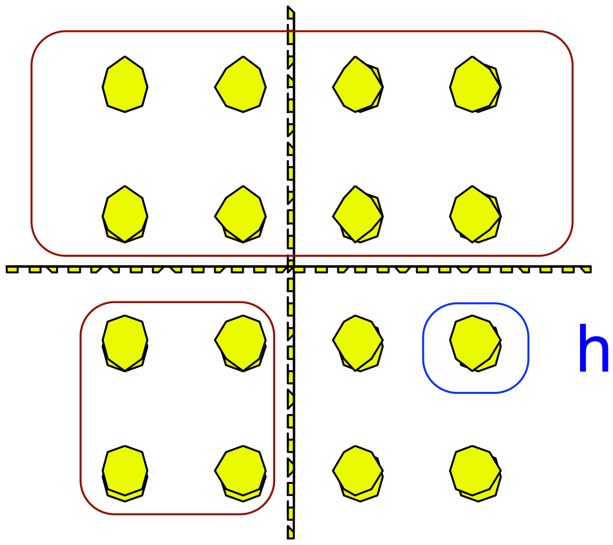


HTA Addressing

h{1,1:2} (hta)

h{2,1}   (array)

hierarchical



h(3,4) ↔ scalar

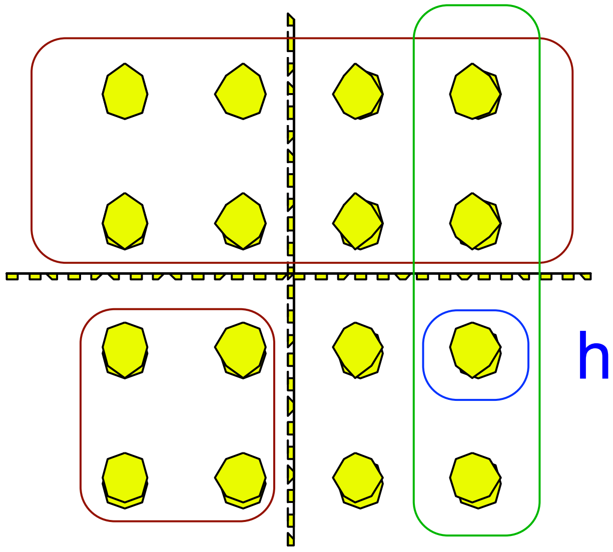
flattened

HTA Addressing

h{1,1:2} (hta)

h{2,1} (array)

hierarchical



h(3,4) ↔ scalar

flattened

h{1:2,2}(1:2,2) ↔ hta

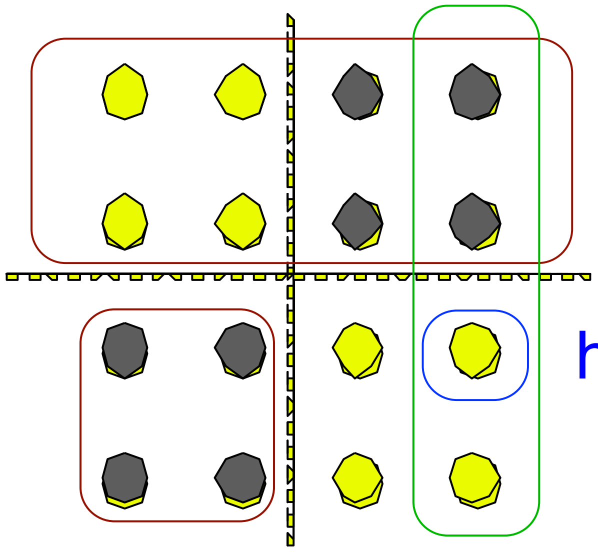
hybrid

HTA Addressing

h{1,1:2} (hta)

h{2,1}   (array)

hierarchical



h{ i + j == 3} ↔ hta

logical indexing

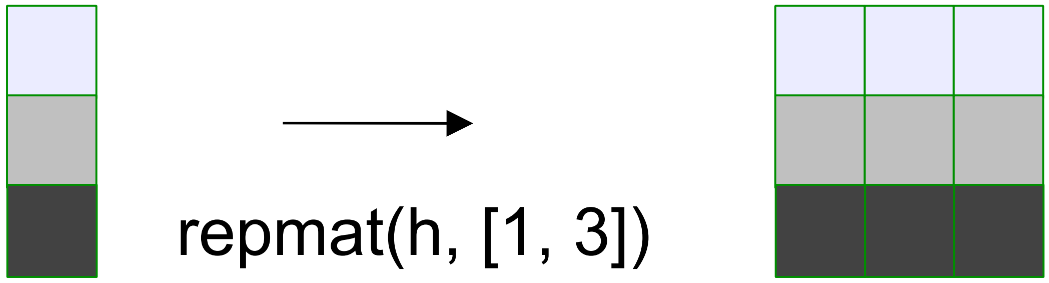
h(3,4) ↔ scalar

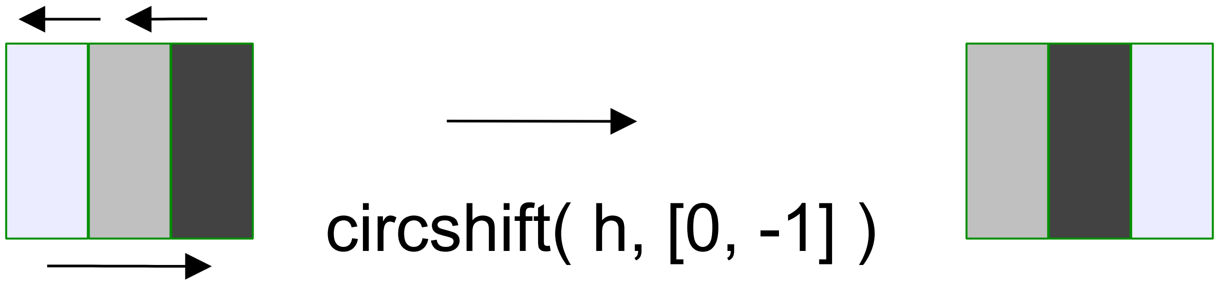
flattened

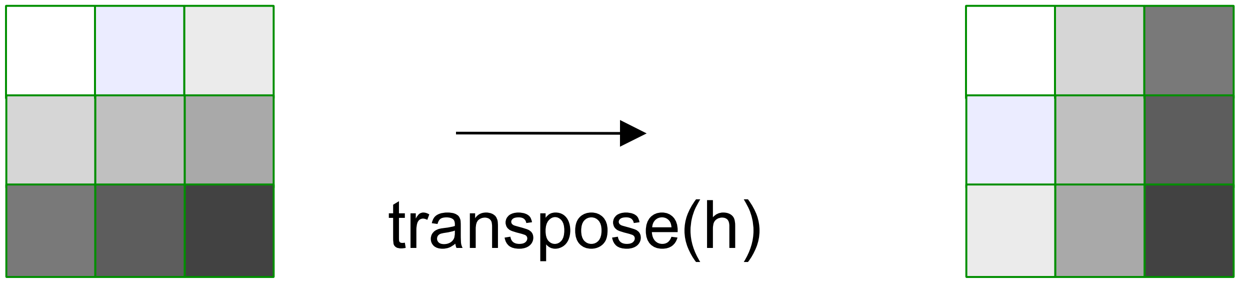
h{1:2,2}(1:2,2) ↔ hta

hybrid

Higher level operations


repmat(h, [1, 3])


circshift(h, [0, -1])


transpose(h)

Higher Level Operations

- Many operators part of the library
  - reduce, circular shift, replicate, transpose, etc

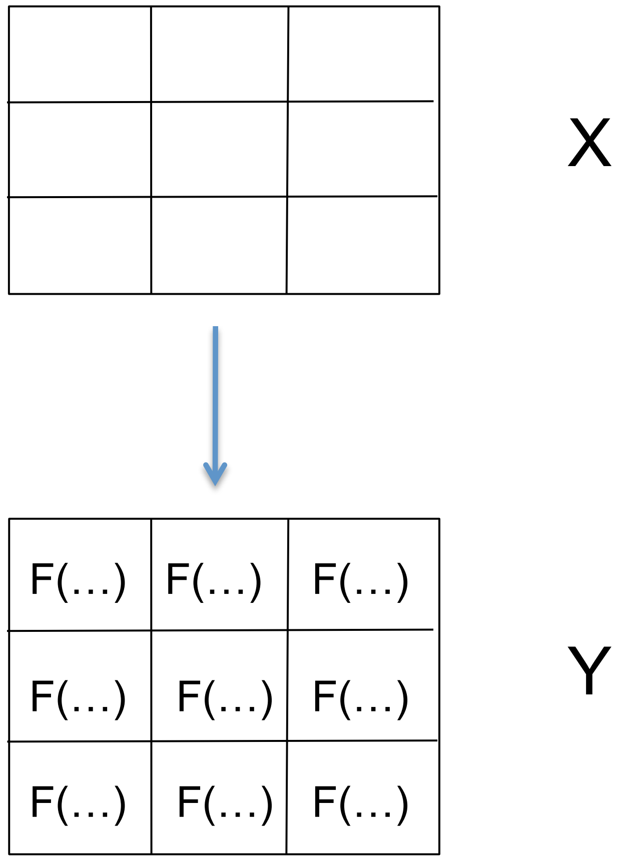
- A map operation (hmap)
  - Applies user defined operators to each tile of the HTA
    - And corresponding tiles if multiple HTAs are passed as input
  - Application of operator occurs in parallel across tiles

User Defined Operations

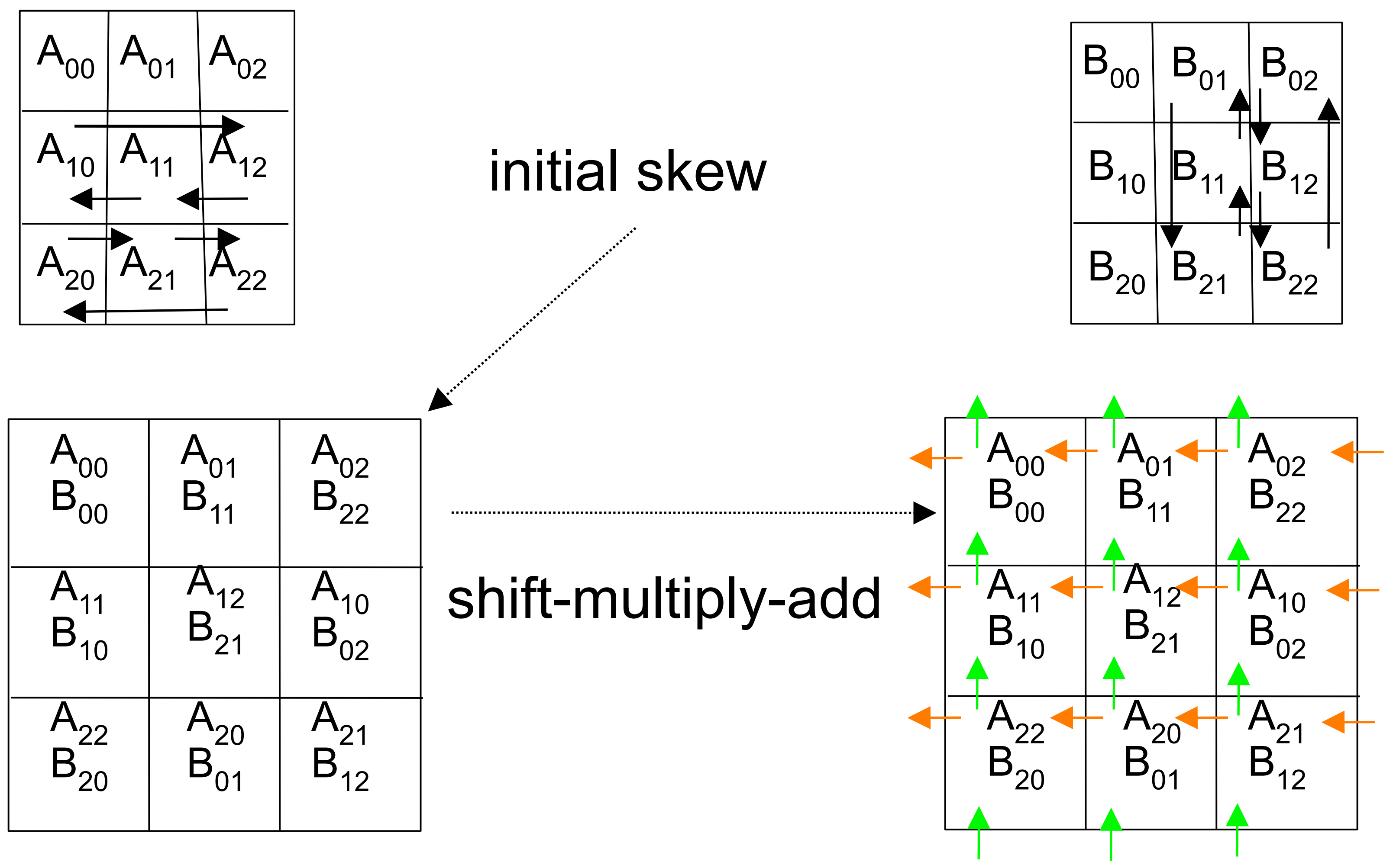
```
HTA X(3,3)[10]
HTA Y(3,3)[10]
…
hmap( F(), X, Y )

F(HTA x, HTA y) {
        y [i] = x[i] * x[i] - 3
}
```

// 3x3 tiles of 10 elements



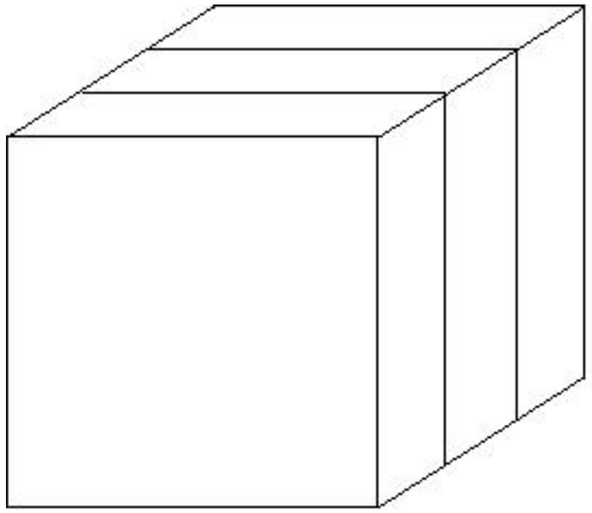
Cannon's Matrix Multiplication


A_{00} A_{01} A_{02}
A_{10} A_{11} A_{12}
A_{20} A_{21} A_{22}
B_{00} B_{01} B_{02}
B_{10} B_{11} B_{12}
B_{20} B_{21} B_{22}
initial skew
A_{00} B_{00} | A_{01} B_{11} | A_{02} B_{22}
A_{11} B_{10} | A_{12} B_{21} | A_{10} B_{02}
A_{22} B_{20} | A_{20} B_{01} | A_{21} B_{12}
shift-multiply-add

Cannon's Matrix Multiplication

```
%Main loop
for i = 1:n
      c = c + a * b;
    a = circshift( a, [0, -1] );
    b = circshift( b, [-1, 0] );
end
```

FT



(a)

```
u = fft (u, [],1);
u = fft (u, [],2);
u = dpermute( u,[3 1 2]);
u = fft( u, [],1);
```

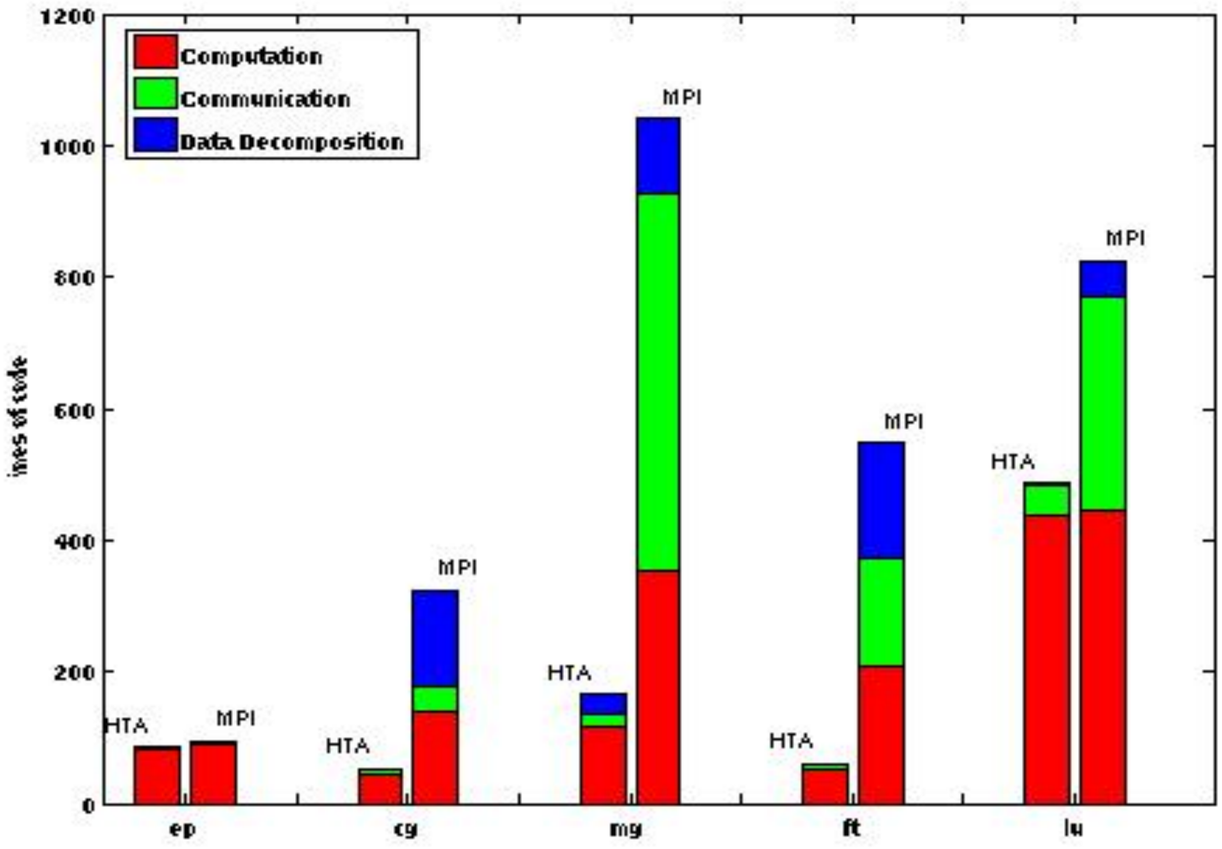
(b)

Advantages of tiling as a first class object for optimization

- HTAs have been implemented as C++ and MATLAB libraries.
  - For shared and distributed memory machines.
- Dense and sparse versions
- Implemented several benchmark suites.
- Performance is competitive with OpenMP, MPI, and TBB counterparts
- Furthermore, the HTA notation produces code more readable than other notations. It significantly reduces number of lines of code.

Advantages of tiling as a first class object

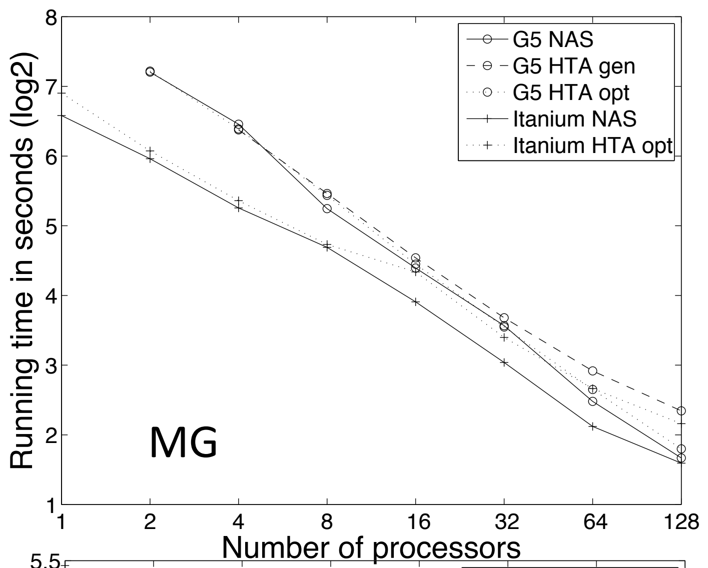
Lines of code

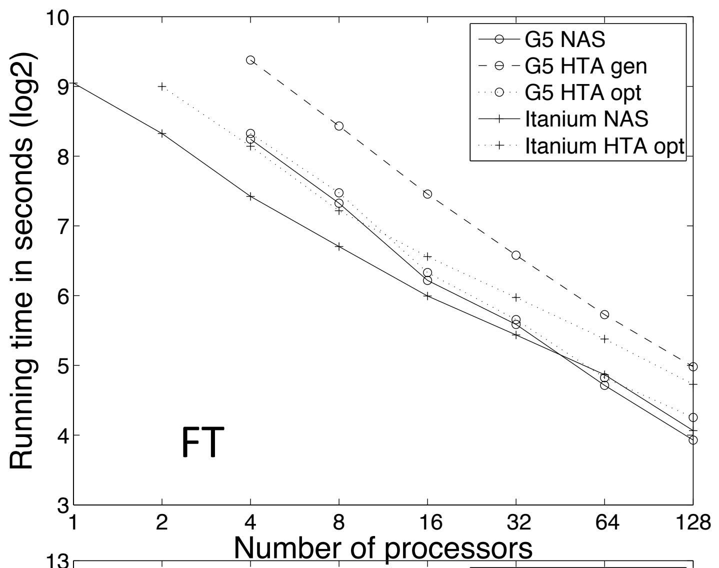


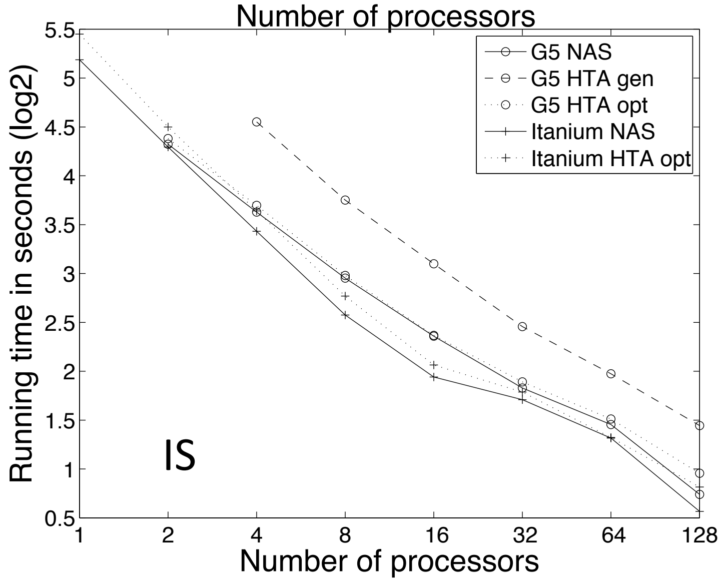
EP CG MG FT LU

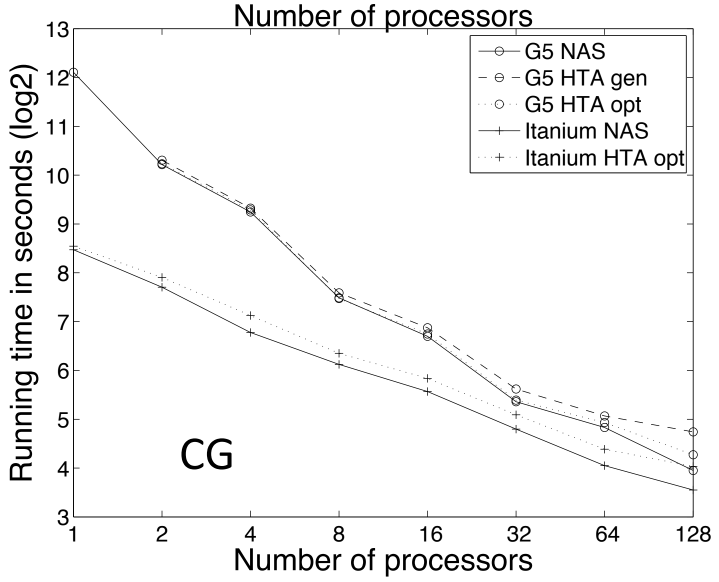
Lines of Code. HTA vs. MPI

Performance Results









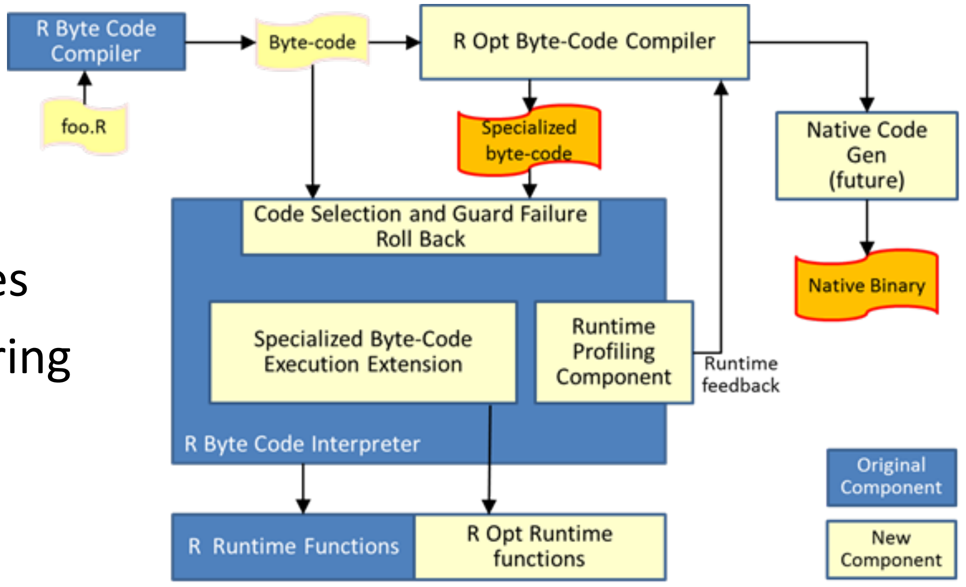
Plans

- We are considering extending ROpt (our extended R interpreter) with HTA operations and enable execution on distributed memory machines.

Work with Peng Wu (IBM Research) and Haichuan Wang (Illinois)

ROpt

- Extends R byte code interpreter using specialization.
  - New Op codes
    - For specific data types
    - For frequently occurring code sequences
  - Simpler data representation
  - Automatic optimization



Preliminary results with ROpt

- On a set of kernels containing mainly scalar operations
  - ROpt delivers average speedup of 3.56 over the bytecode R interpreter
  - ROpt is 12 times slower than C
  - The bytecode interpreter speedup over the original R interpreter is 2.6.
- On the shootout benchmarks
  - 2.07 speedup over the bytecode interpreter but 100 times slower than C.
  - The bytecode interpreter is 2.5 times faster than the original interpreter.