Programming for Parallelism and Locality with Hierarchically Tiled Arrays

Ganesh Bikshandi, Jia Guo, Daniel Hoeflinger
Gheorghe Almasi, Maria Garzaran, Basilio Fraguela,
Christoph von Praun, David Padua
University Of Illinois,Urbana-Champaign,
IBM T. J. Watson Research Center

Problem Statement

- To design abstractions for *parallelism*
- To design abstractions for *locality*
- To measure the abstraction *penality*

Motivation

- Memory Hierarchy is growing
  - Register – Cache (L1, L2) – Memory – Disk – Distributed Memory
- Programming the future systems is a key challenge
  - Portability (Program + Performance)
  - Programmer Productivity
- Automated program transformations infeasible
  - Compiler analysis are limited

State of the Art

- What is wrong with MPI?
  - Low level => lesser productivity
  - SPMD => unstructured codes => bugs
- What is wrong with new Languages?
  - Language => compiler => complexity
  - Learning => development cost
  - Specific => versioning => maintenance cost

Properites

- Required
  - Global View
  - Single threaded
  - Explicit performance control
- Desired
  - Minimal changes to sequential code
  - Minimal set of operations

Overview

- Define a new data type for tiling
  - Recursive data structure
  - A tree-structured representation of physical layout
- Define new operations
  - High-level, F90 like
  - Transparent
- Extend existing languages
  - MATLAB
  - C++

Rationale

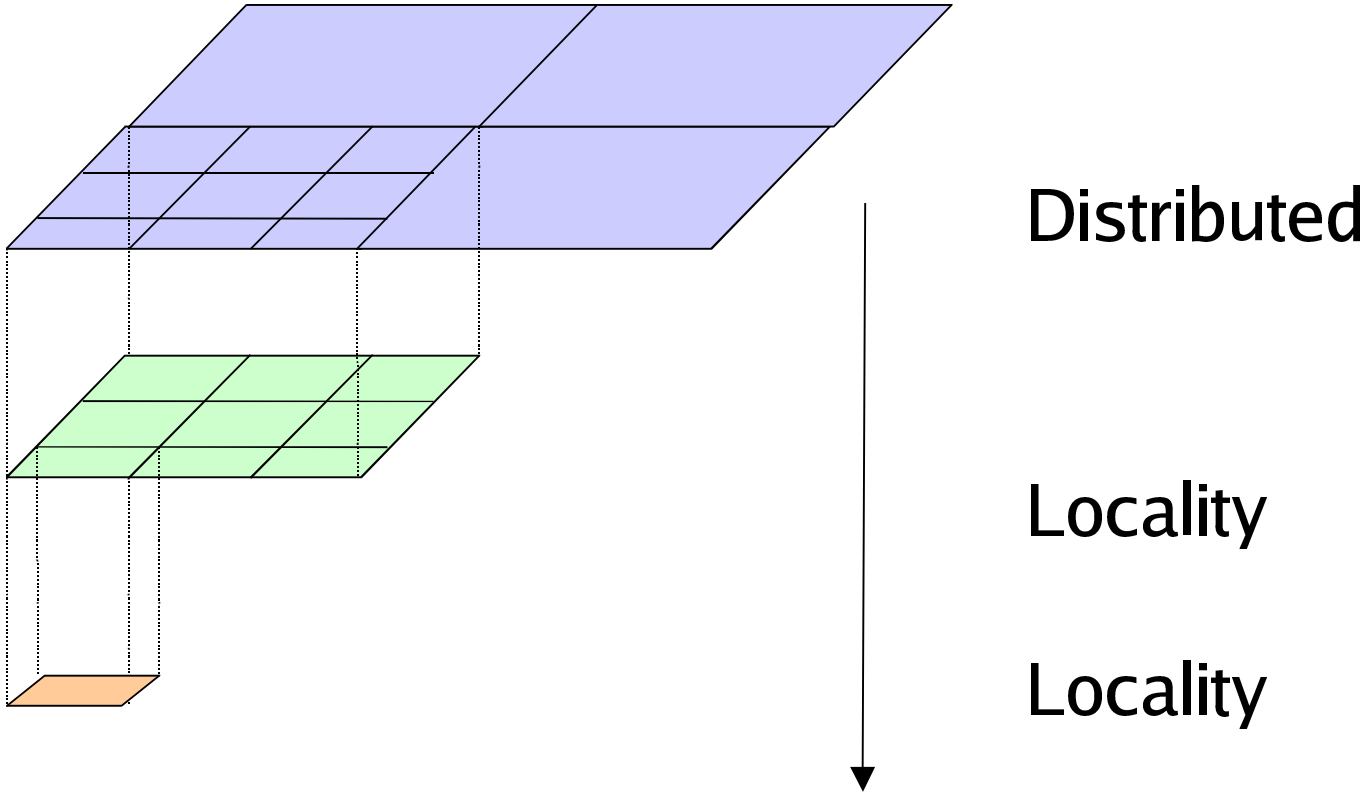
- Tiling is *common* to parallelism and locality
  - Tiling the loops for locality
  - Tiling the data for parallelism (and locality)
- Tiling *benefits* parallelism and locality
  - Message aggregation in MPP
  - Traffic reduction in SMP
  - Locality enhancement in uni-processors
- (TODO: a strong case for parellelism+locality)

Hierarchically Tiled Array

Tiled Array: Array partitioned into tiles such that adjacent tiles have the same size along the dimension of adjacency

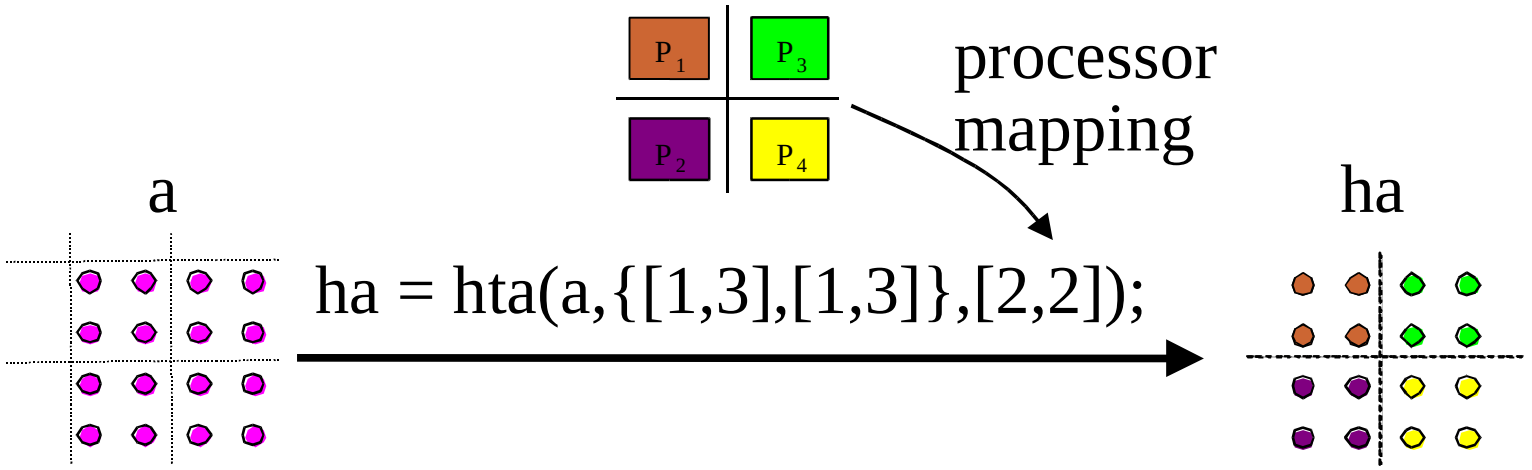
Hierarchically Tiled Array: Tiled Array whose tiles are in-turn tiles

Hierarchically Tiled Array

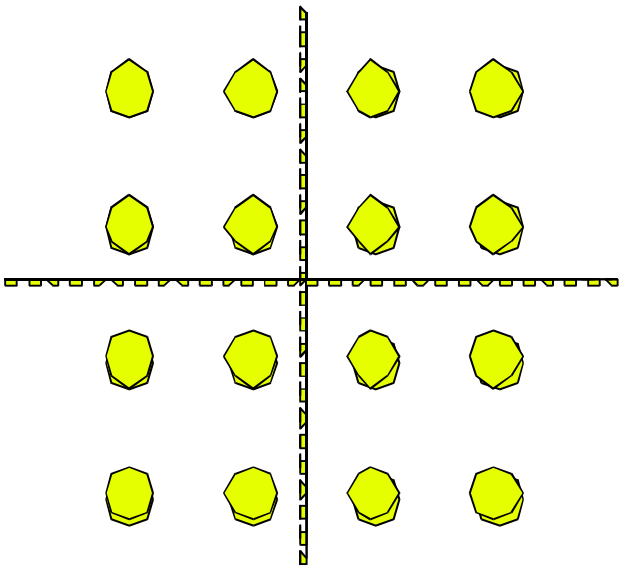


HTA Constructor

hta(MATRIX,{[delim1],[delim2],…},[processor mesh])



HTA Addressing

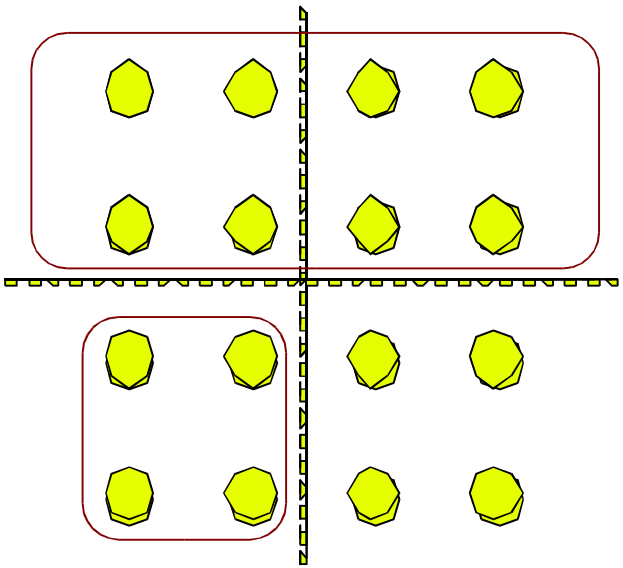


HTA Addressing

h{1,1:2}

h{2,1}

hierarchical

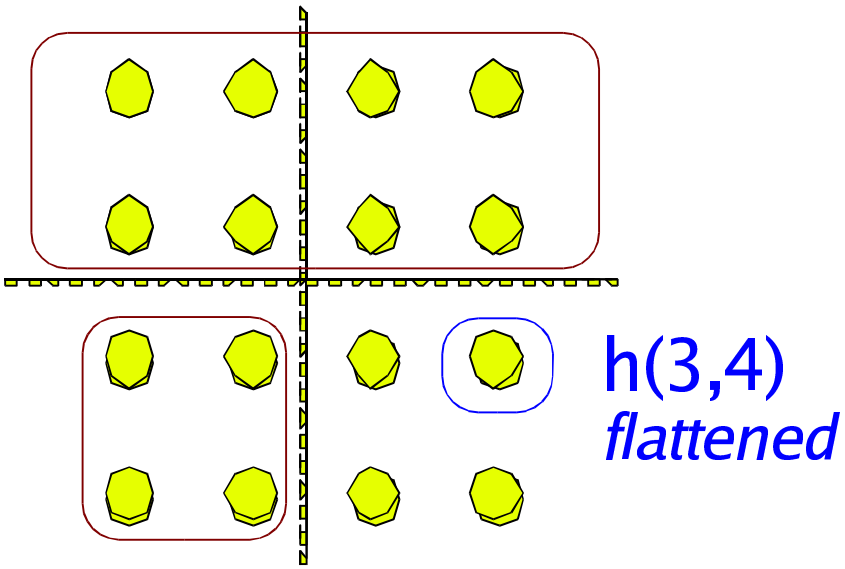


HTA Addressing

h{1,1:2}

h{2,1}

hierarchical



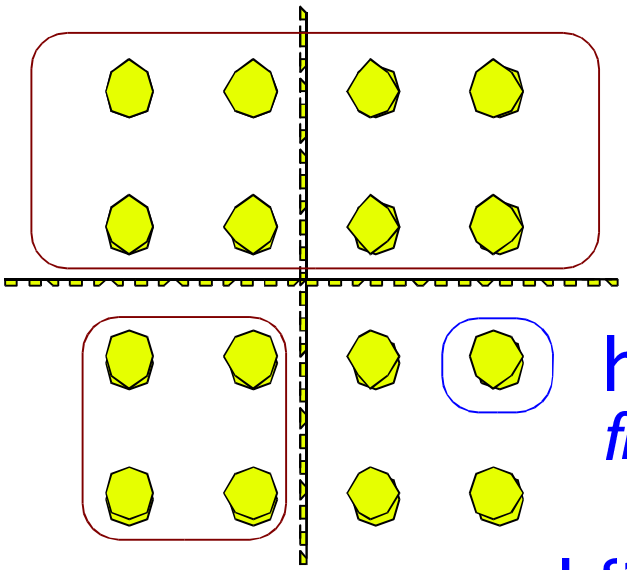
h(3,4)
flattened

HTA Addressing

h{1,1:2}

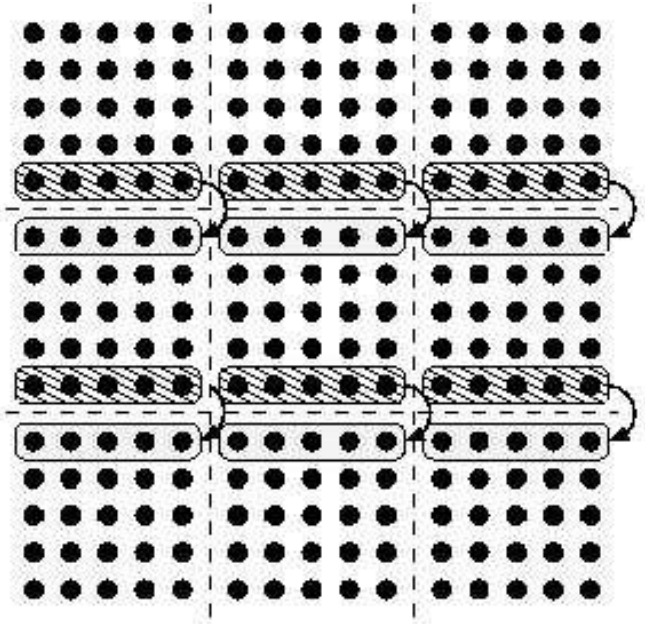
h{2,1}

hierarchical



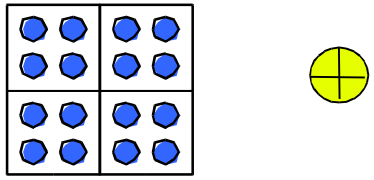
h(3,4)
flattened
or
h{2,2}(1,2)
hybrid

HTA Assignment

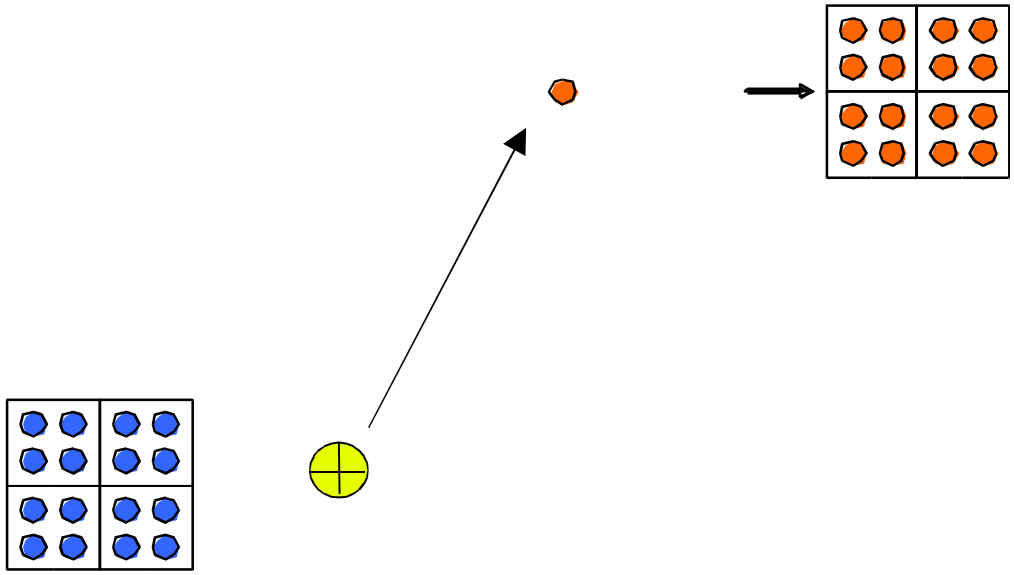


```
V{2:3,:}(1,:)=   V{1:2,:)(5,:)
```

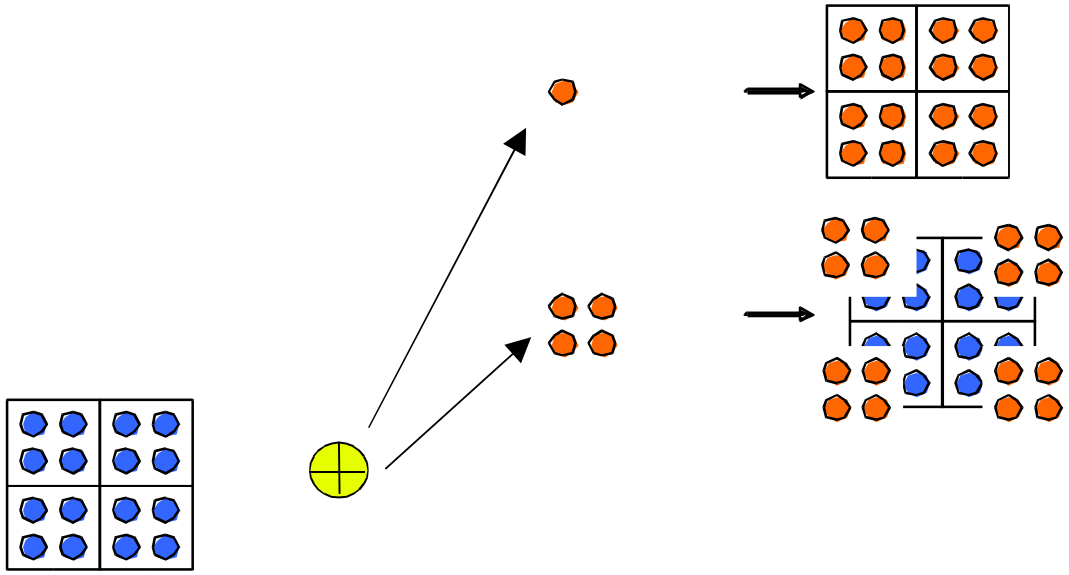
Primitive Operations



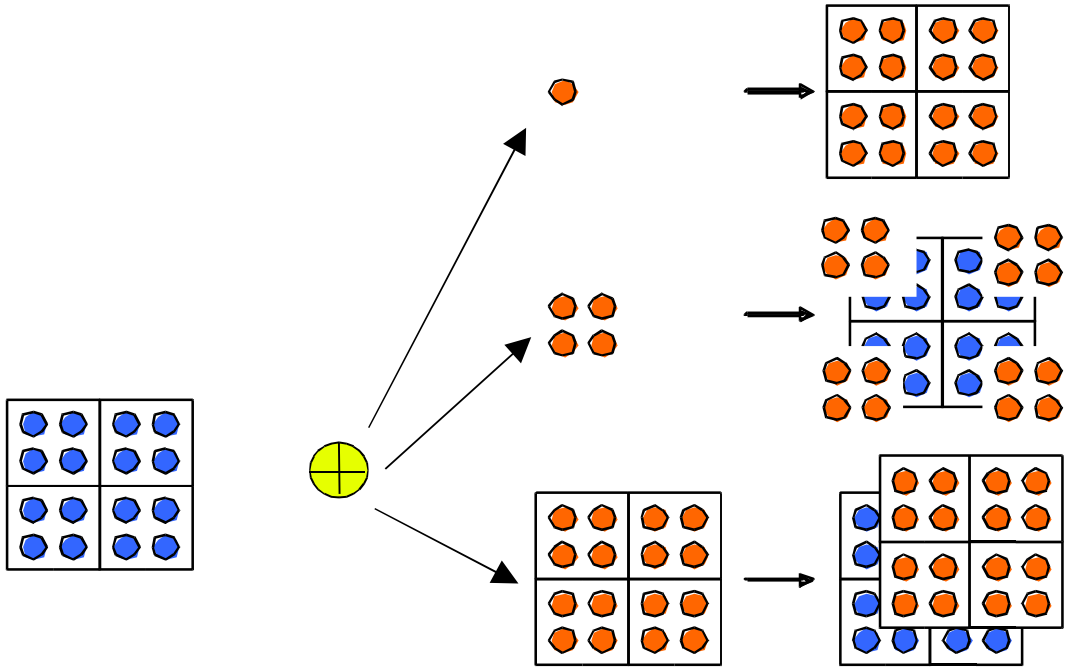
Primitive Operations



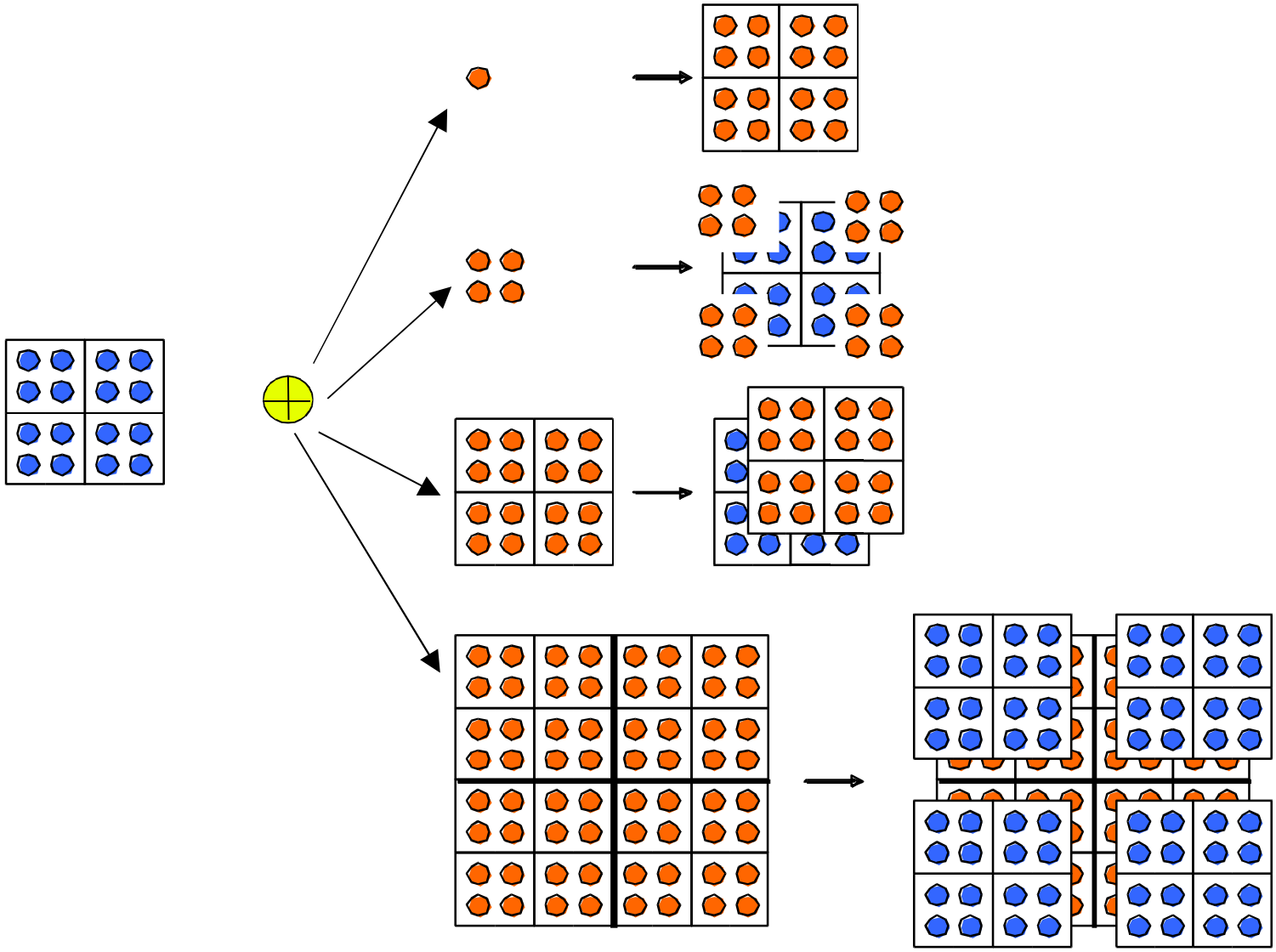
Primitive Operations



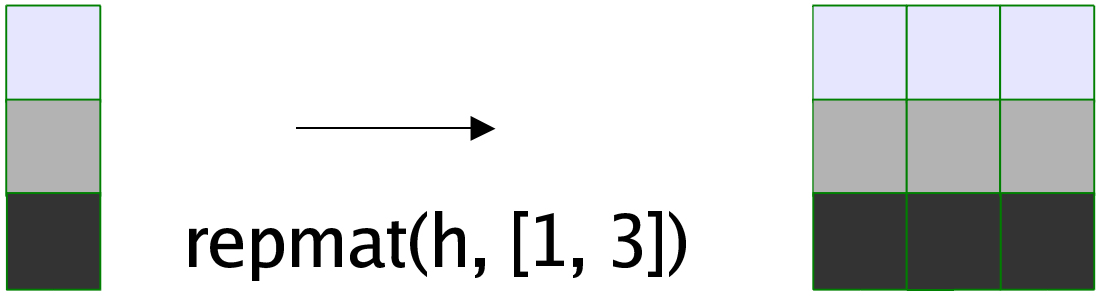
Primitive Operations

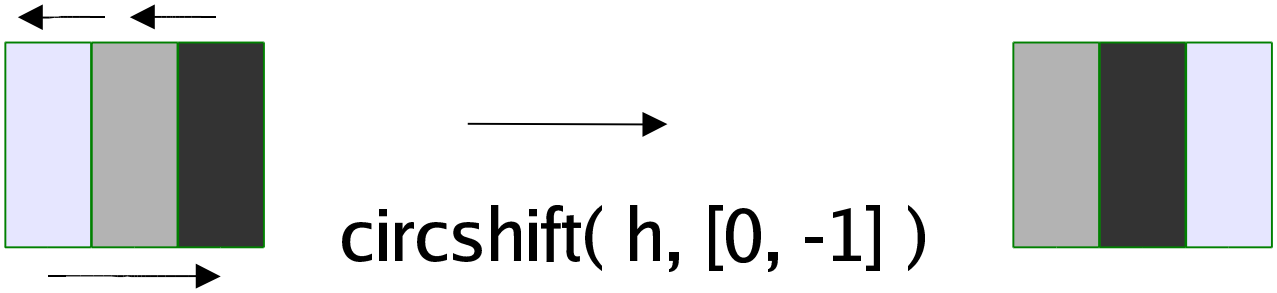


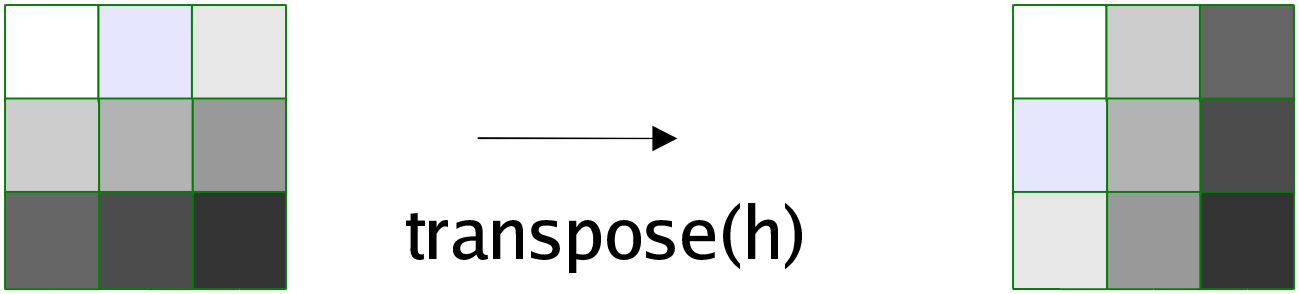
Primitive Operations



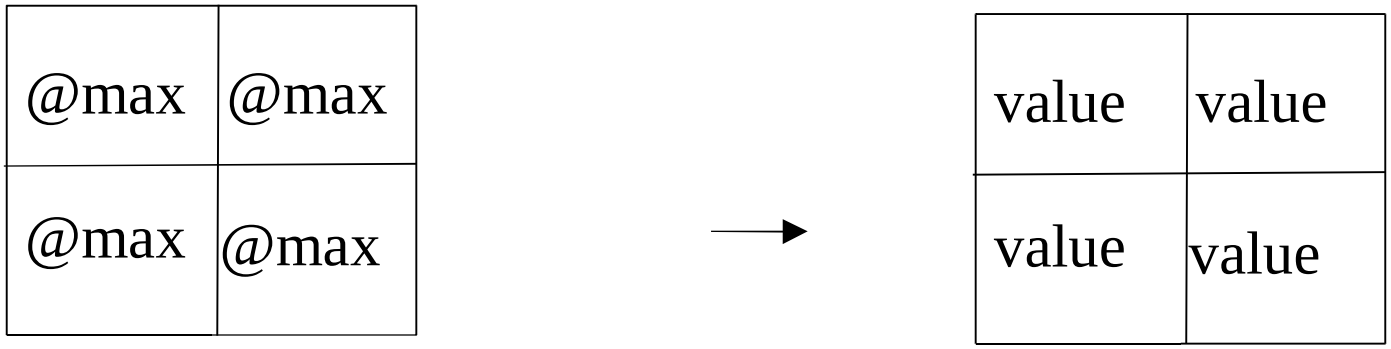
Overloaded Operations

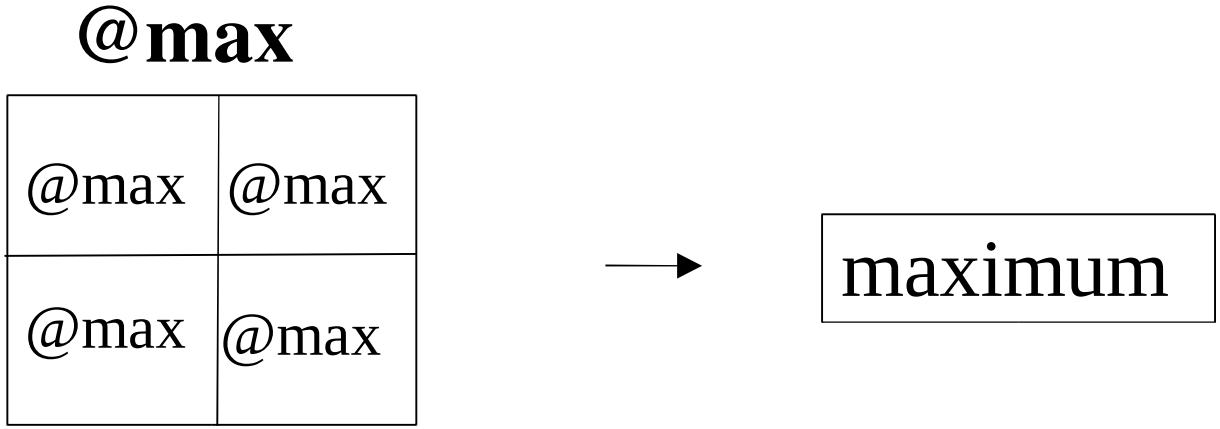






Map/Reduce


@max
@max
@max
@max
value
value
value
value

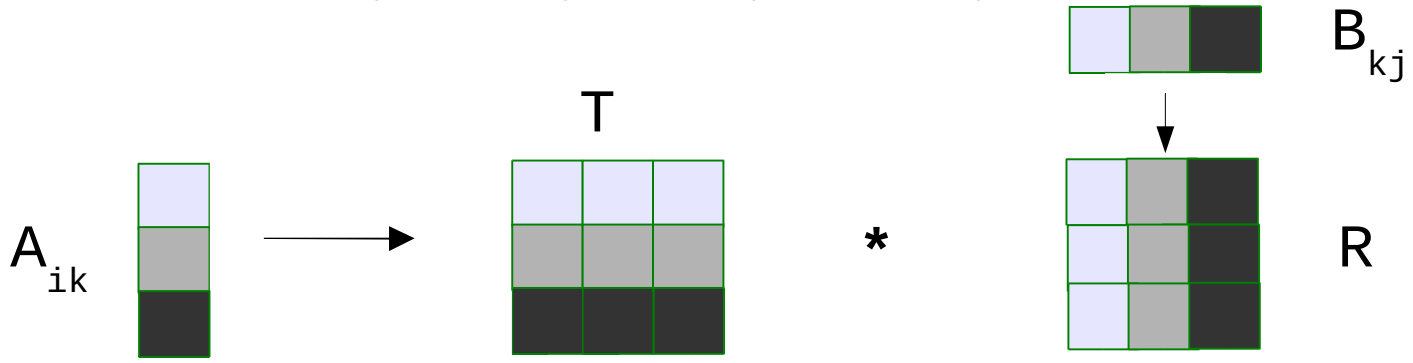

@max
@max
@max
@max
@max
maximum

SUMMA Matrix Multiplication

```
SUMMA-RANK1-MATMUL (A, B, C)
1. C_ij = 0
2. for k = 0 to m-1
3.   T := broadcast A_ik to my row
4.   R := broadcast B_kj to my column
5.   C_ij = C_ij + T_ij .* R_ij
```

SUMMA Matrix Multiplication

```
SUMMA-BLOCK-MATMUL (A, B, C)
1. C_ij = 0
2. for k = 0 to m-1
3.   T := broadcast A_ik to my row
4.   R := broadcast B_kj to my column
5.   C_ij = C_ij + T_ij * R_ij
```



SUMMA Matrix Multiplication

```
function C = summa (A, B, C)
 for k=1:m
    T1 = repmat(A{:, k}, 1, m);
    T2 = repmat(B{k, :}, m, 1);
    C = C + T1 * T2;
 end
```

Blocked Recursive Matrix Multiplication

```
BLOCK-REC-MATMUL (A, B, C)
1. if(size(C) = 1)
2.   then C = C + A * B
3. else
4. for i = 0 to size(A,1)
5.   for j = 0  to size(B,1)
6.     for k = 0 to size(B,2)
7.       BLOCK-REC-MULT( A_ik, B_kj, C_ij)
```

Blocked Recursive Matrix Multiplication

```
template <int LEVEL> void mult(
                    HTA<double, 2, LEVEL> A,
                    HTA<double, 2, LEVEL> B,
                    HTA<double, 2, LEVEL> C) {
  int M = A.shape()[0].size();
  int N = B.shape()[0].size();
  int Q = B.shape()[1].size();
  for (int i = 0; i< M; i++) {
    for (int k = 0; k < N; k++) {
        for (int j = 0; j< Q; j++) {
          mult (A[T(i,k)], B[T(k,j)], C[T(i,j)]);
  }}}}

void mult(double& A,double& B,double& C)
{
  C += A * B;
}
```

NAS Benchmarks

EP – Generating normal deviates

MG – Multi Grid V Cycle

FT – FFT based PDE solver

CG – Inverse Power Iteration + Conjugate Gradient

BT – Block Tri Diagonal Gaussian Elimination

LU – Symmetric-successive over-relaxation

IS – Sorting Integers using bucket sort

NAS Benchmarks

EP – Embarrassingly parallel

MG – Neighbour exchange

FT – Transposes

CG – Reductions

BT – Alternating phases

LU – Linear recurrence

IS – Personalized communication

FT

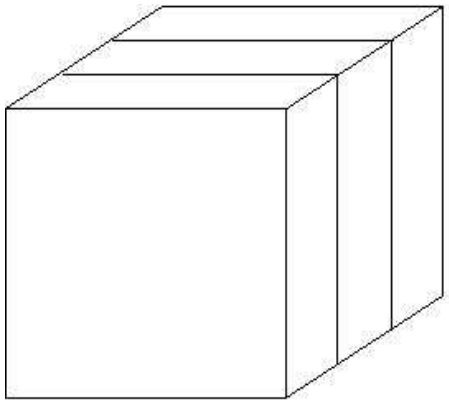
Problem: Solve a Partial Differential Equation

Solution: Fast Fourier Transform

Core: FFT

Pattern: Transpose

FT



(a)

```
u = hta( MX, {1, 1, 1:nz/pz:nz},[1 1 pz]);
u = fft (u, [],1);
u = fft (u, [],2);
u = dpermute( u,[3 1 2]);
u = fft( u, [],1);
```

(b)

CG

Problem: Solve Au = v.

Solution: Conjugate Gradient

Core: Sparse Matrix Vector Multiplication

Pattern: Global Reductions

CG – Outline

$r = x$

$\rho = x^T r$

$p = r$

DO i=1,25

$q = A * p$

$\alpha = r / p^T q$

$z = z + \alpha * p$

$\rho_0 = \rho$

$r = r - \alpha * q$

$\rho = r^T * r$

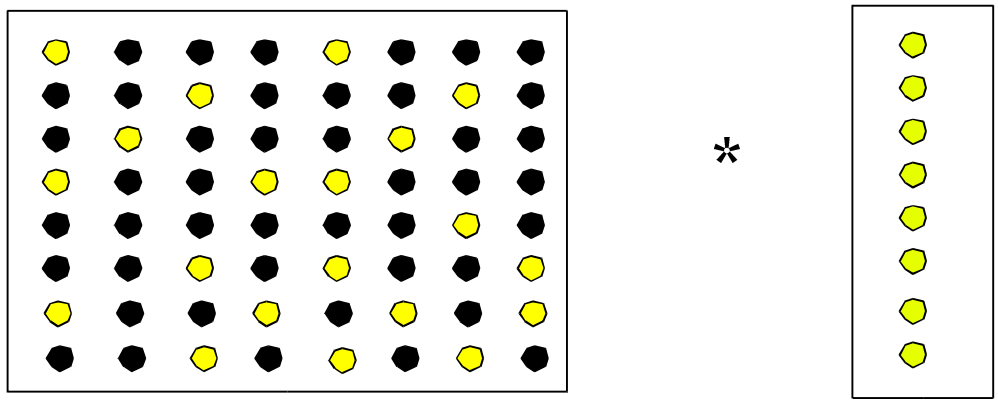
$\beta = \rho / \rho_0$

$p = r + \beta * p$

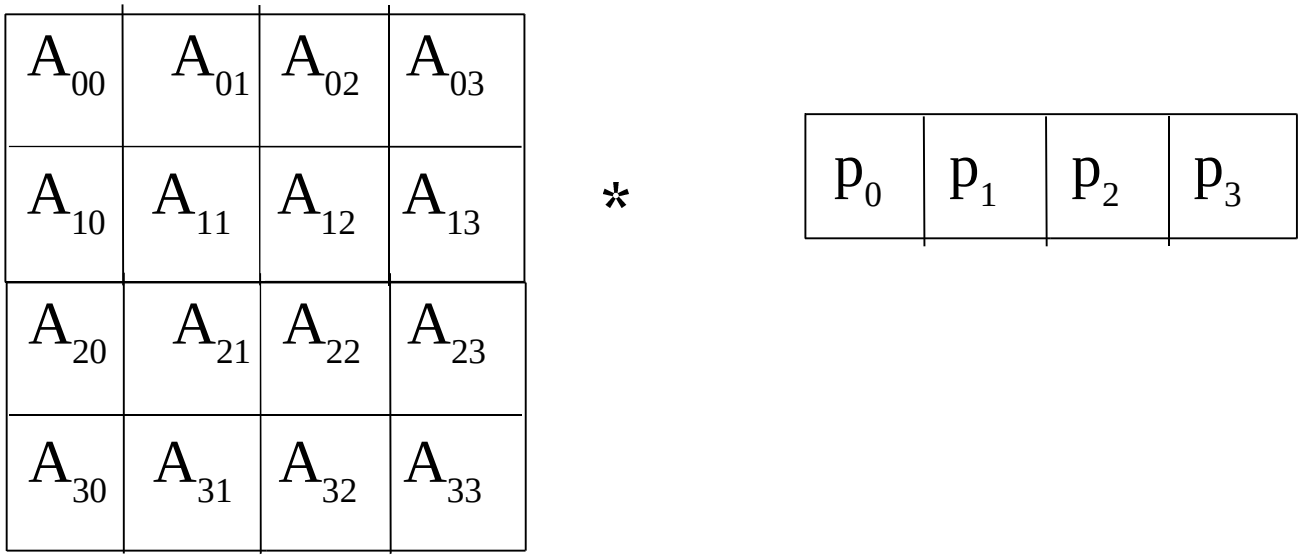
ENDDO

return $r = |x - Az|$

Sparse Matrix -Vector Multiplication

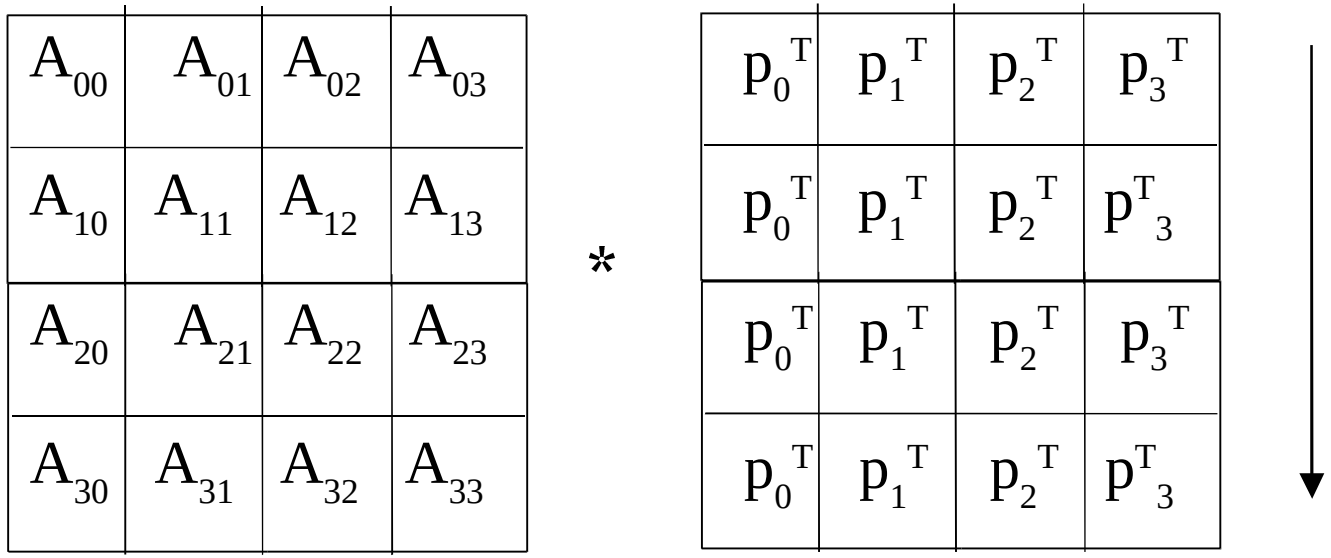


Sparse Matrix -Vector Multiplication



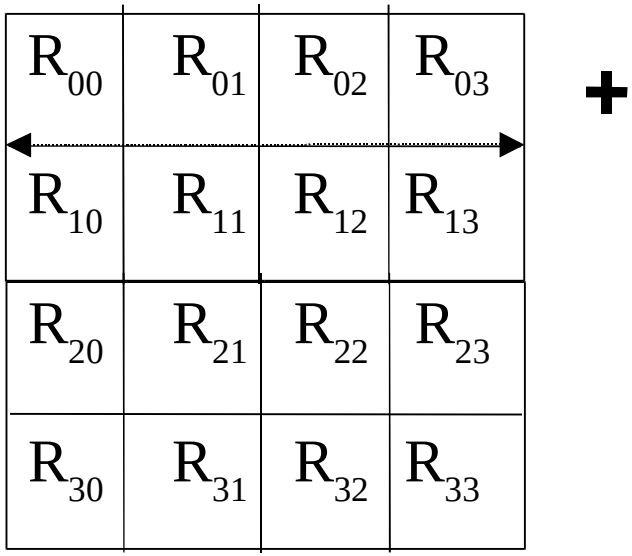

```
A=hta(MX,{partition_A},[M N]);
p = hta( vector', {partition_p}[1 N] );
```

Sparse Matrix -Vector Multiplication




```
p = repmat(p, [M 1]);
p = ltrans(p);
```

Sparse Matrix -Vector Multiplication




```
q = reduceHTA( 'sum', mult(a, p), 2,
true);
```

Sparse Matrix -Vector Multiplication

```
%create the  HTA
a=hta(MX,{partition_A},[M N]);

%distribute, replicate and prepare vector p
p = repmat( hta( vector', {partition_p} [1 N] ),[M,1]);
p = ltrans(p)

%multiply and reduce
q = reduceHTA( 'sum', mult(a, p), 2, true);
```

Scalability Challenges

- MATLAB
  - Interpretation
  - Vectorization
  - Copy-on-write
  - Poor data abstraction (e.g. LU and BT)
- C++
  - Run tim e costs (e.g. Virtual functions)
  - Extensible design
  - Syntax

Scalability Challenges

- Eliminating temporaries
  - e.g. stencil convolutions
- Scalable communication
  - e.g. reductions
  - e.g. Transpose

Experimental Results

?????

Related Works

- Languages
  - ZPL, CAF, UPC, HPF, X10, CHAPEL
- Parallel MATLABs excluded
- Classification
  - Global vs Segmented memory
  - Single vs Multiple thread of execution
  - Language vs Library

Result

Approach	Implementation		Address Space	Control
	Language	Library		
CAF	√		Global	MT
GAS		√	Global	MT
HPF	√		Global	ST
HTA		√	Global	ST
MPI/PVM		√	Local	MT
POET		√	Global	MT
POOMA		√	Global	ST
Titanium	√		Global	MT
UPC	√		Global	MT
X10	√		Global	MT
ZPL	√		Global	MT

```
 void(conj_grad (SparseMatrix A, Matrix x, Matrix z) {
  z = 0.0;
  q = 0.0;      /* Conjugate Gradient Benchmark in C++ */
  r = x;
  p = r;
  rho = Matrix::dotproduct(r, r);
  for(int i = 0; i < CGITMAX; i++) {
    qt = 0;
    pt = p.ltranspose();
    SparseMatrix::lmatmul(A, pt, &qt);
    qt = qt.reduce(plus<double>(), 0.0, 1, 0x2);
    q = qt.transpose();
    alpha = rho / Matrix::dotproduct(p, q);
    z = z + alpha * p;
    rho0 = rho;
    r = r - alpha * q;
    rho = Matrix::dotproduct(r, r);
    beta = rho / rho0;
    p = r + beta * p;
  }
  qt = 0;   // compute the norm
  Matrix zt = z.ltranspose();
  SparseMatrix::lmatmul(A, zt, &qt);
  qt = qt.reduce(plus<double>(), 0.0, 1, 0x2);
  r = qt.transpose();
  Matrix f = x - r;
  f = f * f;
  norm = f.reduce(plus<double>(), 0.0);
  rnorm = sqrt(norm);
}
```

Conclusion

- HTA is a new data abstraction
  - F90 like syntax
  - Explicitness
- Unification of
  - Parallelism & locality
  - Sequential & parallel
- Future work
  - C++ library + NAS benchmarks (under progress)
  - Stronger cases for Hierarchical tiling