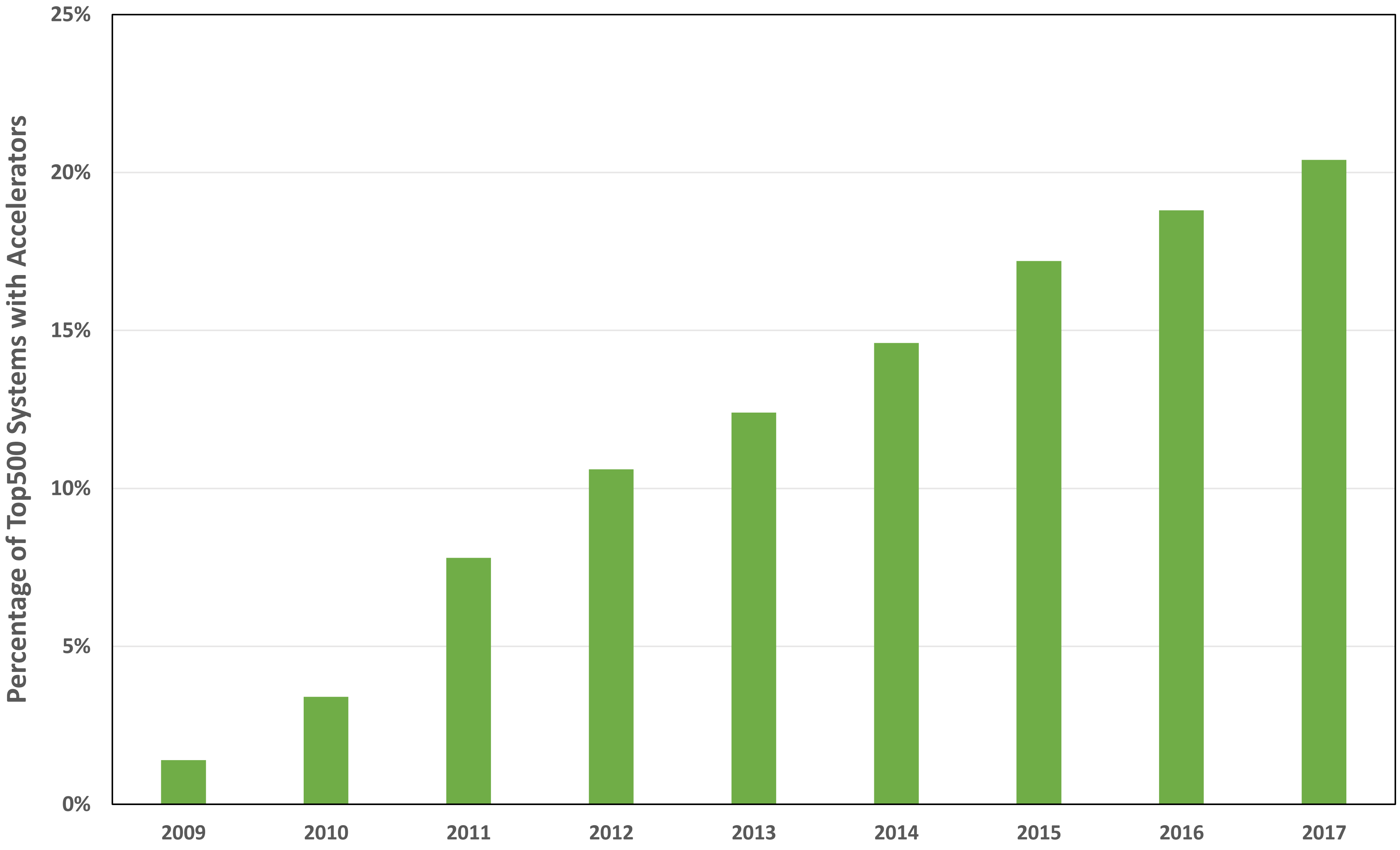
Investigating Data Layout Transformations in Chapel

APAN QASEM
TEXAS STATE UNIVERSITY

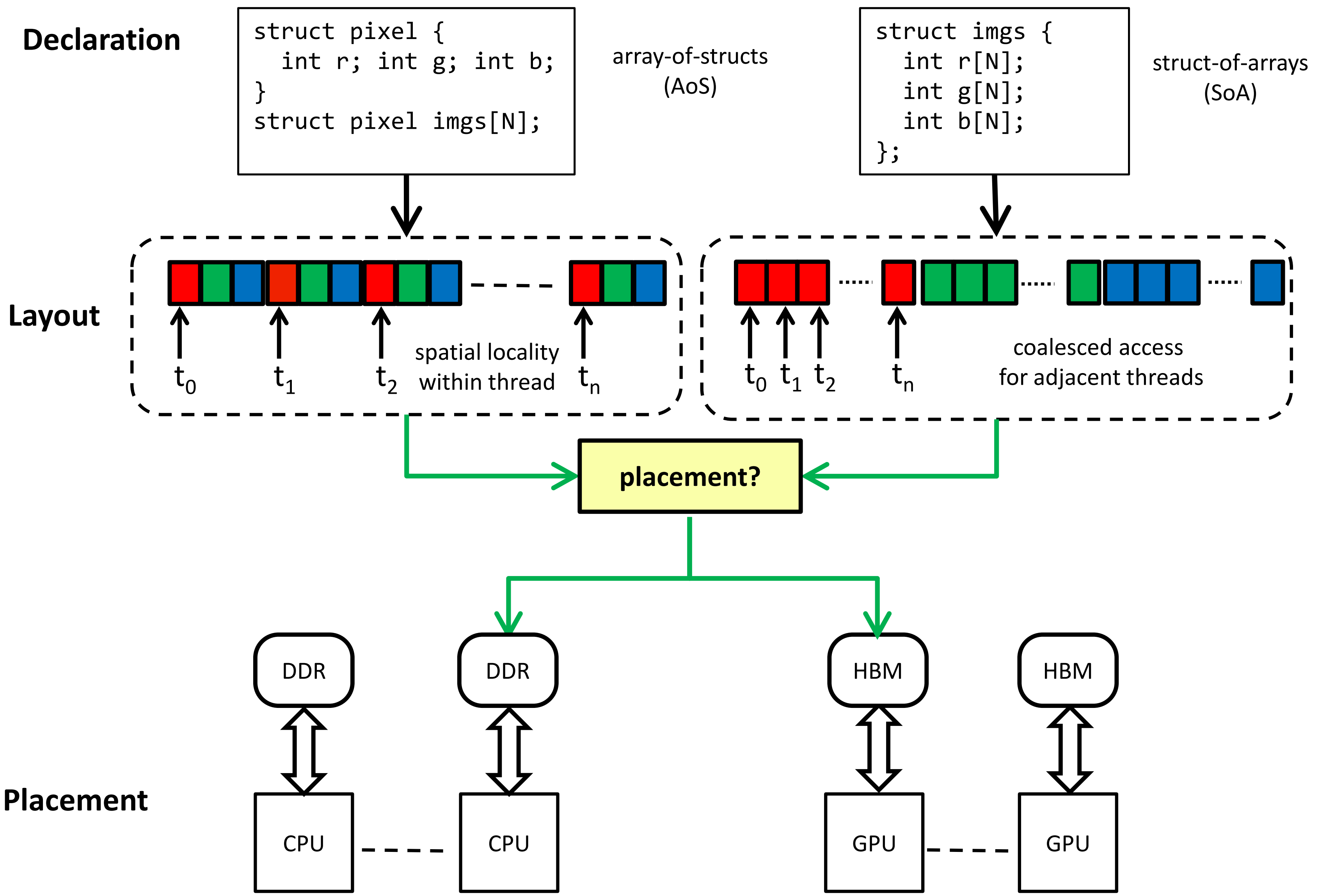
ASHWIN M. AJI, MICHAEL L. CHU
AMD RESEARCH

CHIUW 2018

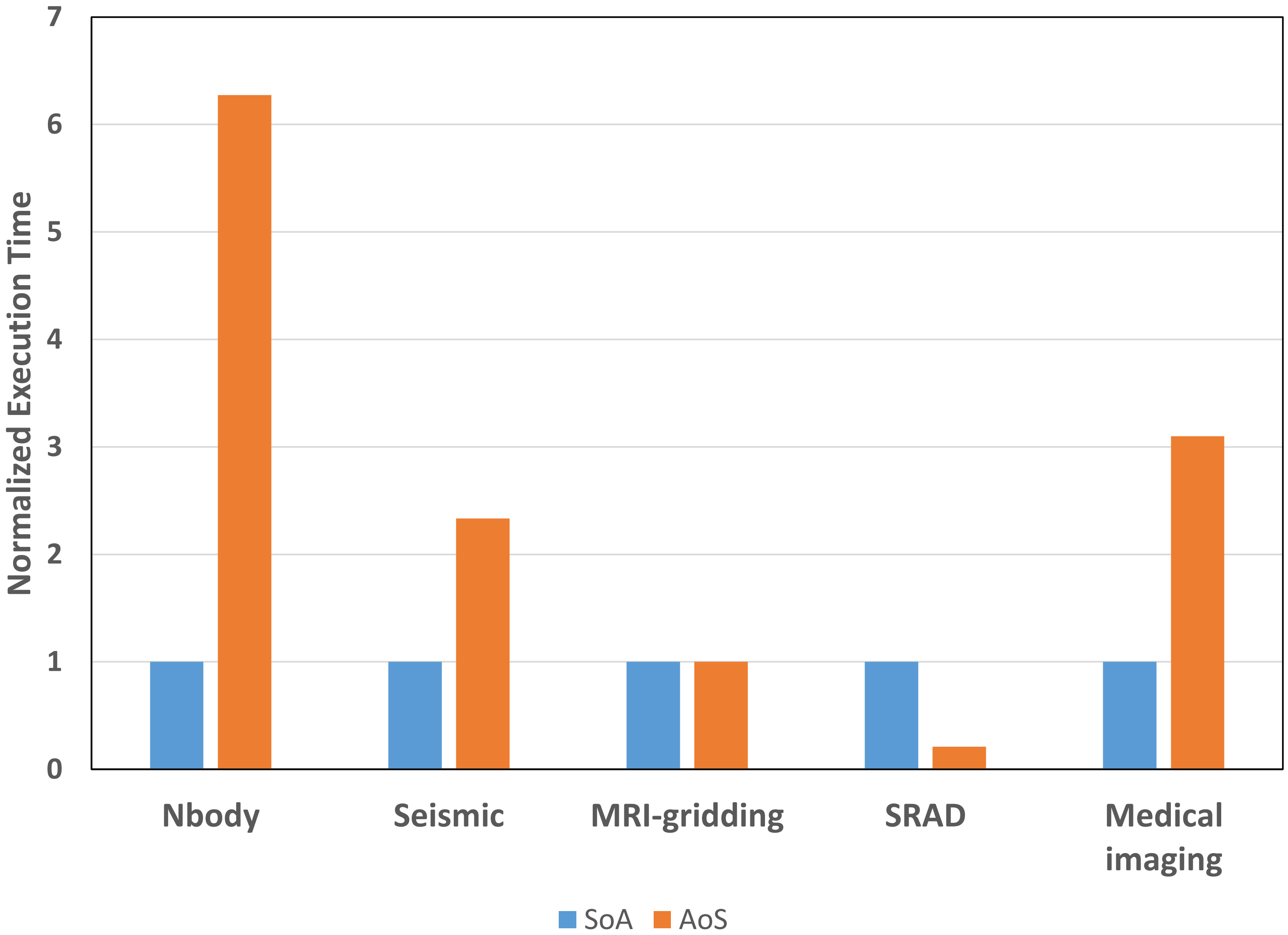
EMERGENCE OF HETEROGENEOUS NODE ARCHITECTURES



THE DATA ORGANIZATION PROBLEM



PERFORMANCE IMPACT OF DATA LAYOUT



OTHER EXAMPLES OF DATA LAYOUT ISSUES

AMD

- Access to non-contiguous array locations

```
void conv2D(float** A, float** B) {
  int i, j;
  float c11, c12, c13;
  c11 = +0.2;  c21 = +0.5;  c31 = -0.8;
  ...
  for (i = 1; i < NI - 1; ++i) {
    for (j = 1; j < NJ - 1; ++j) {
      B[i][j] = c11 * A[i - 1][j - 1] +
                c12 * A[i][j - 1]  +
                c13 * A[i + 1][j - 1]
      ...
}
```

Serial C

```
__global__ void conv2D(float *A, float *B) {

  int i = blockIdx.x * blockDim.x + threadIdx.x;
  float c11, c12, c13;
  c11 = +0.2;  c21 = +0.5;  c31 = -0.8;
  int j;
  for (j = 0; j < NJ - 1; j++ )
   B[i * NJ + j] = c11 * A[(i-1) * NJ + (j-1)] +
                   c12 * A[i * NJ + (j-1)] +
                   c13 * A[(i+1) * NJ + (j-1)]
   ...
}
```

Parallel CUDA

OTHER EXAMPLES OF DATA LAYOUT ISSUES

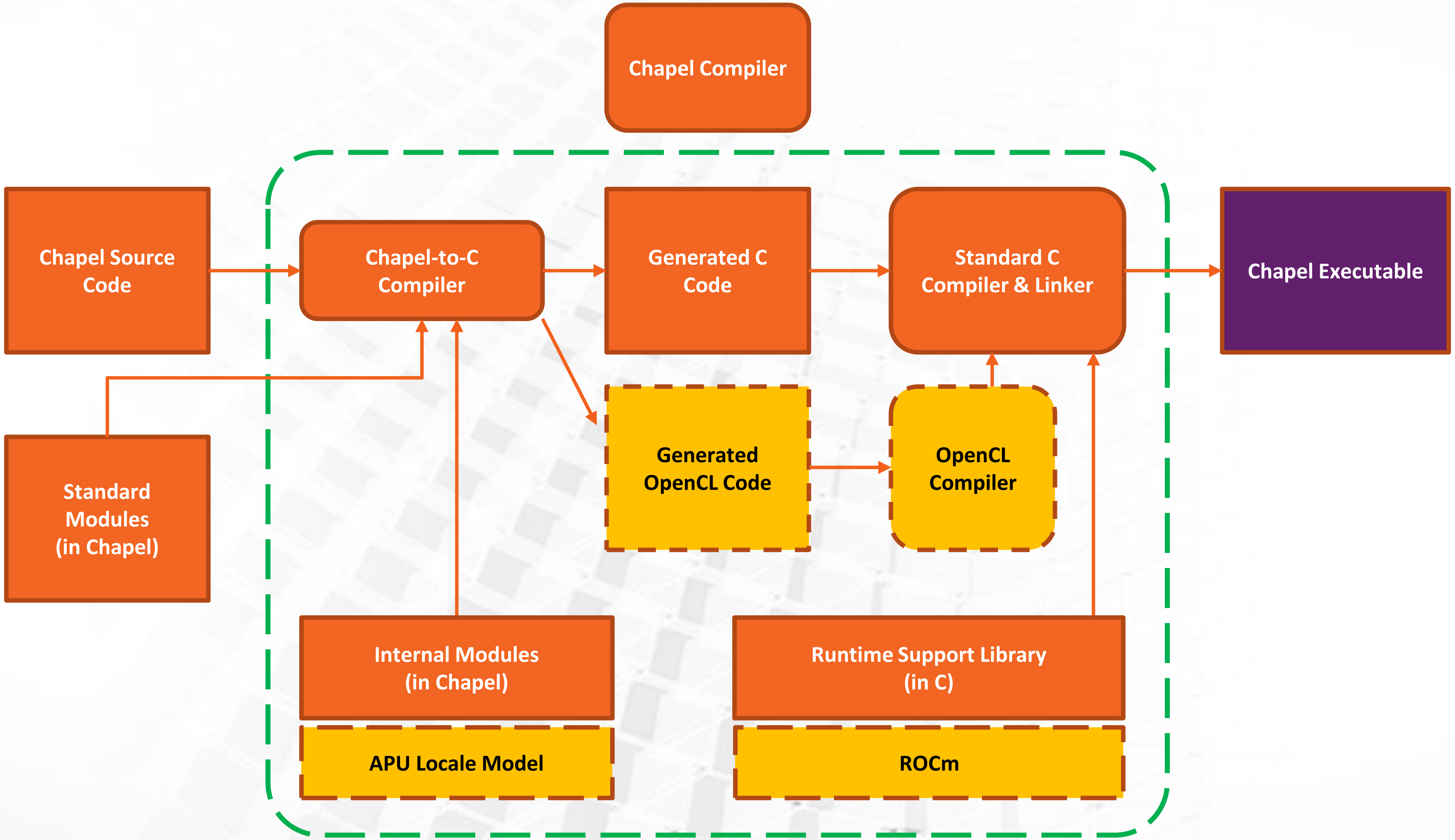
- Sparse Data Access
  - Non-unit stride access to data structure

```
mb_sad_calc(unsigned short *blk_sad,
            unsigned short *frame, ... ) {
    int tx = f(threadIdx.x)
    int ty = f(threadIdx.x);
    ...
    int cur_o = f(tx, ty, width, …);
    ...
    for(int y = 0; y < 4; y++)
     for (int x = 0; x < 4; x++)
       sad4x4 += abs(ty, tx, x, y, ..)
              - frame[cur_o + y * width + x]);
}
```

THIS WORK

- Investigates mechanisms for data layout transformations in Chapel
- Leverages Chapel-specific features
  - Index sets
  - Domain maps
- Explores both automatic and semi-automatic transformations
  - No change to the core language (as yet)

CHAPEL COMPILATION FRAMEWORK WITH GPU EXTENSIONS



DATA LAYOUT TRANSFORMATION 1: COLUMN-MAJOR TRANSFORMATION

AMD

```
var domRowMajor : domain(2) = {rows, cols};
var domColMajor : domain(2) dmapped ColMajor() = {rows, cols};

var A : [domRowMajor] real ;
var B : [domColMajor] real ;

for i in rows {
   for j in cols {
       A[i,j] = 17.0; // accessed as row-major
       B[i,j] = 0.0; // accessed as col-major
   }
}
```

- Implemented ColMajor as a domain map
- Modeled after DefaultRectangular module
  - Constructor redefinition: remap index sets
  - Adjustment of dsiAccess() methods

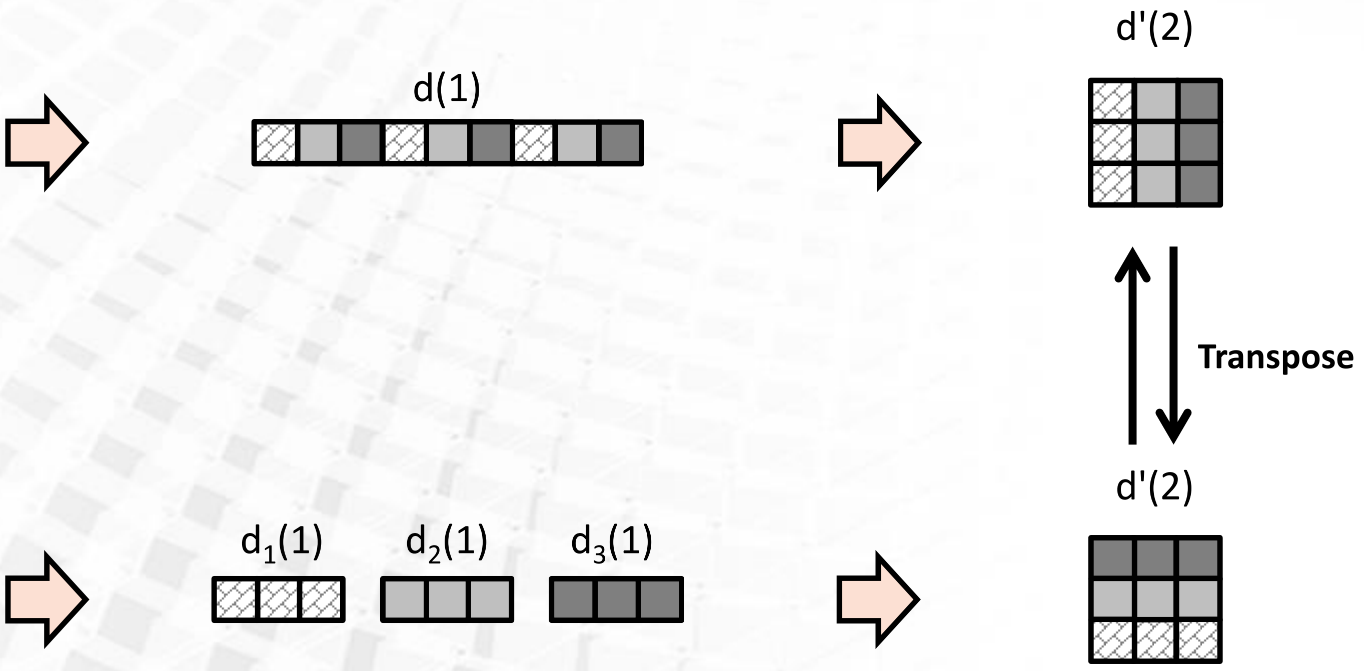
DATA LAYOUT TRANSFORMATION 2: AOS-TO-SOA TRANSFORMATION

AoS

```
record aos {
  var x: real;
  var y: real;
  var z: real;
}

var D: domain(1) = 1..3;
var A: [D] aos;
```

SoA

```
var N = 1..3;
record soa {
  var x : [N] real;
  var y : [N] real;
  var z : [N] real;
}

var A  = new soa();
```



Declaration | **Domain Representation** | **Transformed 2D domain**

- Represent AoS as a rectangular domain
- Transpose domain to get from AoS to SoA

EXAMPLE AOS TO SOA IN CHAPEL

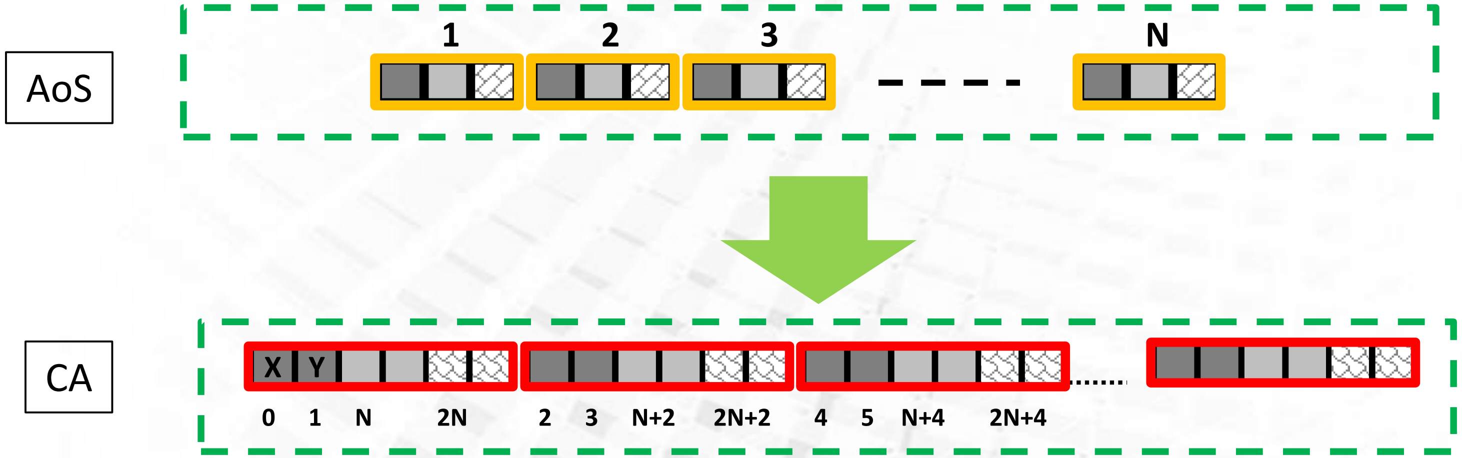
```
var D: domain(1) = 1 .. N;
record pixel {
  var r: real;
  var g: real;
  var b: real;
}

var src : [D] pixel;
var dst : [D] pixel;

on Locales[0].GPU do {
  forall i in 1..N {
    dst[i].r = src[i].r * v0 – src[i].g * v1;
    ...
}
```

Before

```
use LayoutTransposed;   // layout transform module

record pixel {
  var r: real; var g: real; var b: real;
}

var DomSoA : domain(2) dmapped Transposed() = {1..3, 1..N};

var src : [DomSoA] real;   // array of element type
var dst : [DomSoA] real;   // array of element type

on Locales[0].GPU do {
   forall i in 1..N
     dst[1,i] = src[1,i] * v0 - src[2,i] * v1;
     ...
}
```

After

DATA LAYOUT TRANSFORMATION 3: COMPRESSED ARRAY LAYOUT



- Fields within an AoS are coalesced to reduced memory divergence and improve register usage
- Tiling for improved cache locality and local memory usage
- Repositioning to handle sparse access of data
- All three done within the same Compressed Array (CA) domain map

EXAMPLE CA TRANSFORMATION IN CHAPEL

```
var D: domain(1) = 1 .. N;
record pixel {
  var r: real;
  var g: real;
  var b: real;
}

var src : [D] pixel;
var dst : [D] pixel;

on (Locales[0]:LocaleModel).GPU do {
  forall i in 1 .. N {
    dst[i].r = src[i].r * v0 – src[i].g * v1;
    ...
}
```

Before

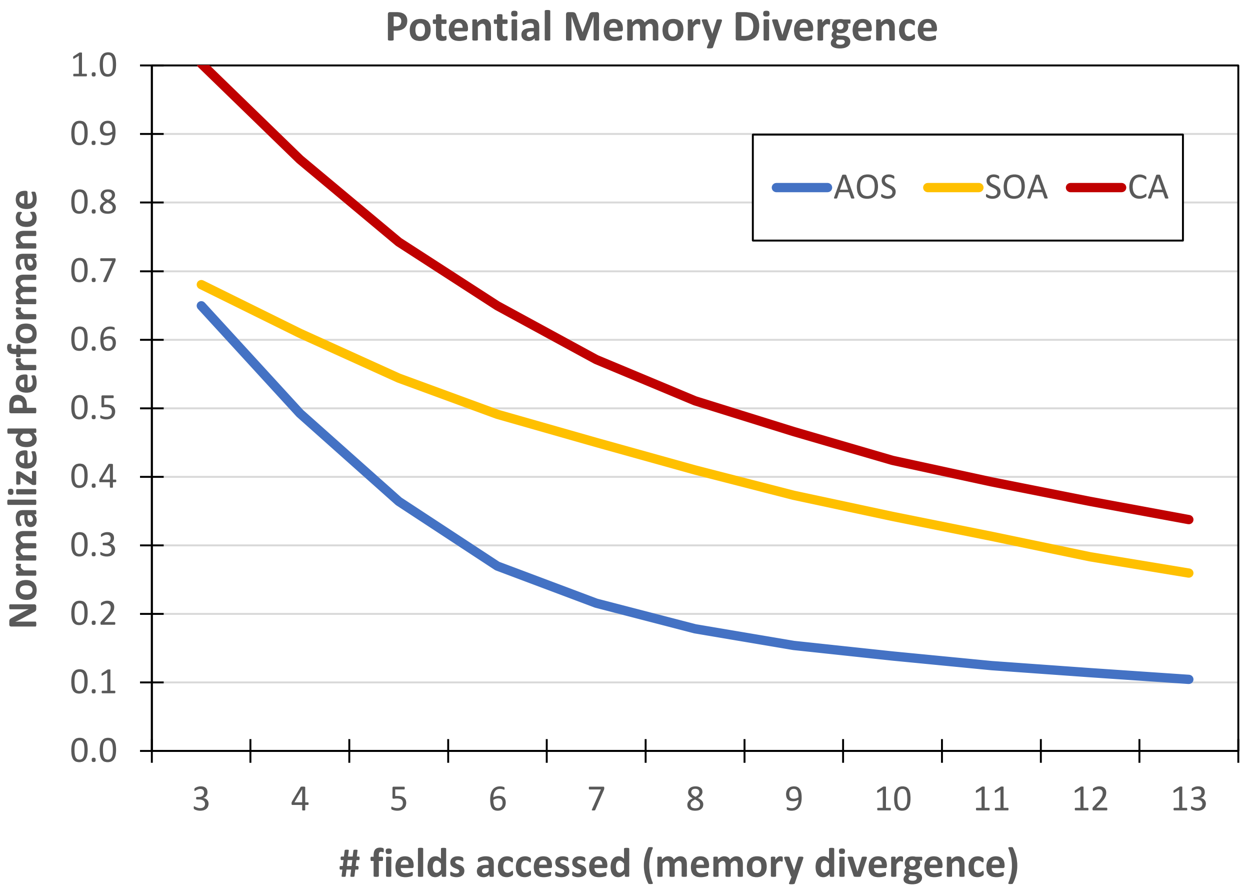
```
use LayoutCA;   // layout transform module

record pixel {
  var r: real; var g: real; var b: real;
}

param t = getTileSize();   // source code analysis
param s = getSparsity();   // source code analysis

var DomCA : domain(2) dmapped CA(t,s) = {1..3, 1..N};

var src : [DomCA] real;   // array of element type
var dst : [DomCA] real;   // array of element type

on (Locales[0]:LocaleModel).GPU do {
   forall i in 1..N
     dst[1,i] = src[1,i] * v0 - src[2,i] * v1;
     ...
}
```

After

IMPLICATIONS OF DATA LAYOUT ON MEMORY DIVERGENCE

AMD

- Platform:
  - AMD A10-8700B APU
- Running synthetic micro-kernels tunable for
  - Arithmetic intensity
  - Problems size
  - Data access patterns
- Potential Memory Divergence (PMD) = estimate of expected memory divergence based on access patterns
- Performance normalized to CA at PMD of 3 (higher is better)



CONCLUSIONS AND FUTURE WORK

- We demonstrate that the domain map feature of Chapel can be useful in implementing sophisticated data layout transformations
  - Column Major
  - AoS to SoA
  - Compressed Array

- Future work
  - Explore fully automatic approaches for data layout transformation
  - Explore collaborative design patterns
  - Extend work to discrete GPU nodes

AMD

DISCLAIMER & ATTRIBUTION

ATTRIBUTION