
Leveraging Hardware Address Sampling

Beyond Data Collection and Attribution

Xu Liu

Department of Computer Science

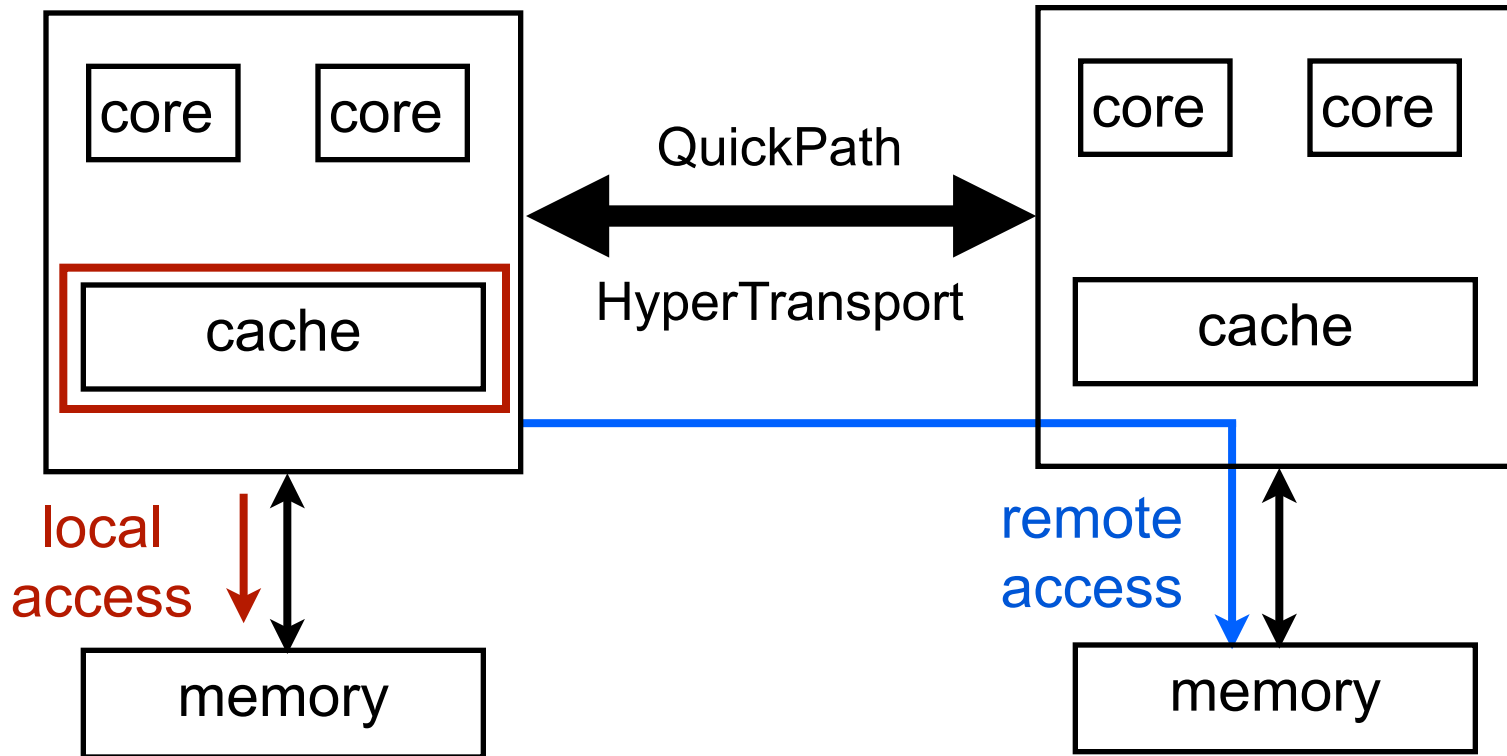
College of William and Mary

xl10@cs.wm.edu



Motivation: Memory is the Bottleneck

NUMA: Non-Uniform Memory Access





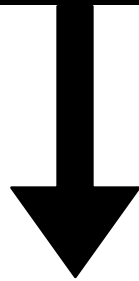
State of the Arts

simulation methods

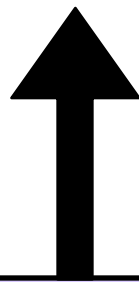
deep insights

weaknesses:

- 2-5x overhead
- not real machines



low overhead with deep insights



deep insights with low overhead

measurement methods

low overhead



Hardware Address Sampling

- Features of address sampling
 - sample memory-related events (memory accesses, NUMA events)
 - capture effective addresses
 - record precise IP of sampled instructions or events
- Support in modern processors
 - AMD Opteron 10h and above: instruction-based sampling (IBS)
 - IBM POWER 5 and above: marked event sampling (MRK)
 - Intel Itanium 2: data event address register sampling (DEAR)
 - Intel Pentium 4 and above: precise event based sampling (PEBS)
 - Intel Nehalem and above: PEBS with load latency (PEBS-LL)
- Efficient memory measurement (SC'13)
 - code-centric analysis
 - data-centric analysis



Code-centric vs. Data-centric

- Code-centric attribution
 - problematic code sections
 - instruction, loop, function

```
1: #pragma omp parallel for num_threads(4)
2: for (i = 0; i < n; i++) {
3:   for(j = 0; j < n; j++) {
4:     for(k = 0; k < n; k++) {
5:       A[i, j, k] = A[i, j, k] + B[j, i, k] + C[k, j, i];
6:     }
7:   }
8: }
```

code-centric profiling

line 5: 100% latency

data-centric profiling

array A:

line 5: 1% latency

array B

line 5: 10% latency

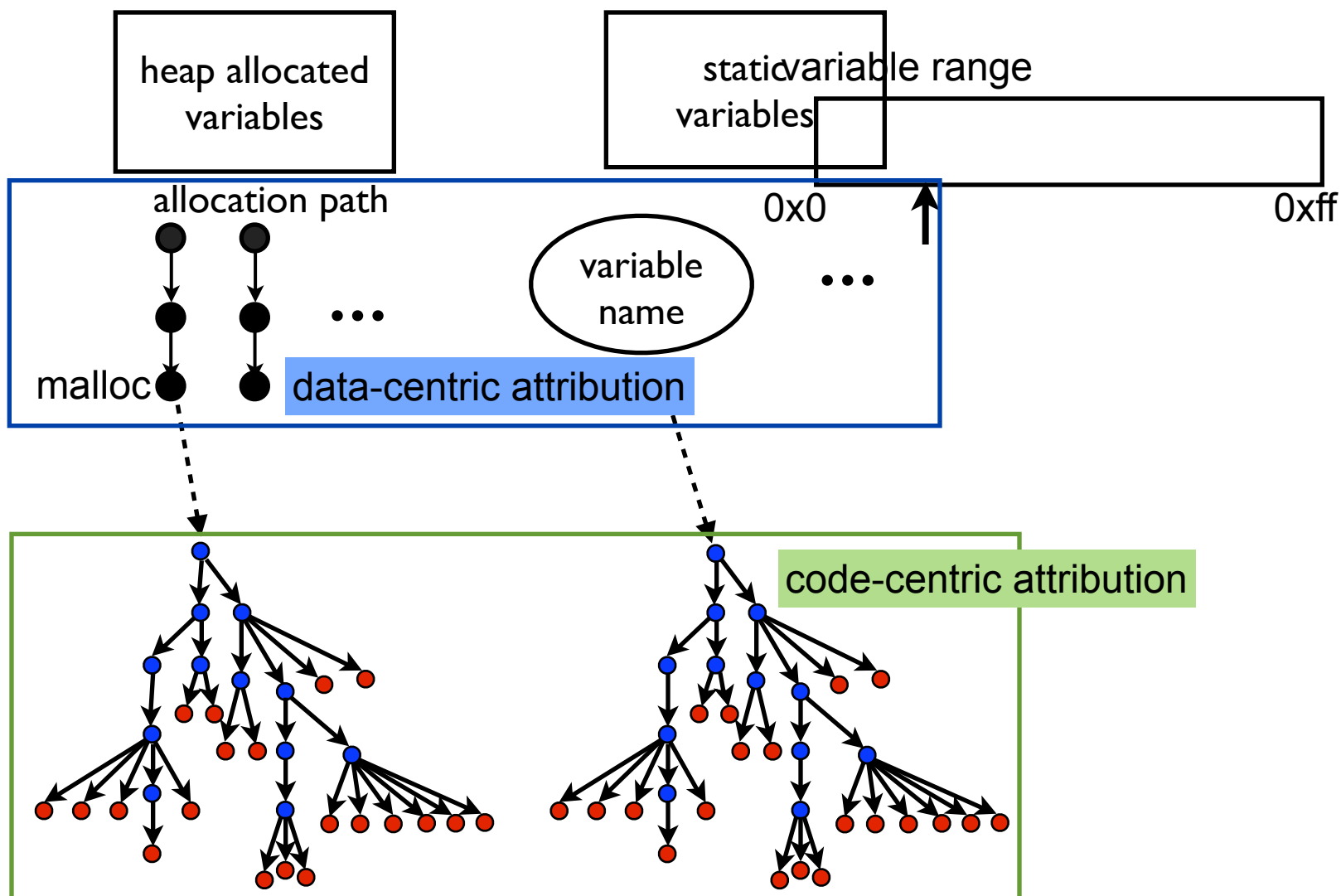
array C

line 5: 89% latency

Combining code-centric and data-centric attribution
provides additional insight

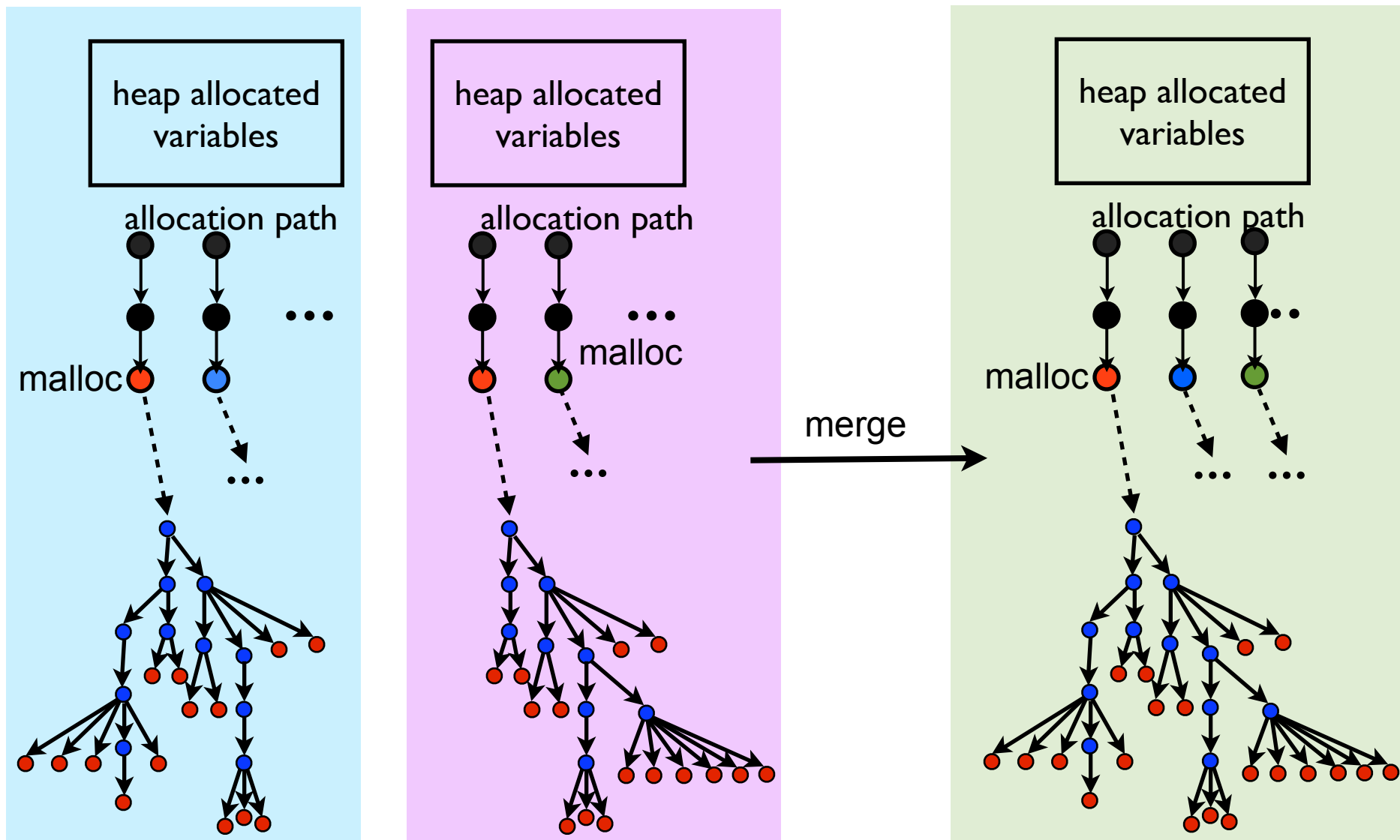


Attributing Samples





Aggregating Profiles





LULESH on Platform of 8 NUMA Domains

Visual Studio Code editor showing the LULESH.3D.noGlobal.TALC.C.bak.C source file. The code defines arrays for coordinates and velocities. A red arrow points from the `*z = new Real_t[numNode];` line to the 'call site of allocation' in the 'Scope' view.

Calling Context View: Callers View, Flat View

Scope: Experiment Aggregate Metrics, monitored_heap_data, 267: heap_data_allocation, 297: monitor_main, 479: main, 2160: operator new[](unsigned long), 32: operator new(unsigned long), 52: malloc, heap_data accesses

R_DRAM_ACCESS:Sum (l)

Scope	R_DRAM_ACCESS:Sum (l)
Experiment Aggregate Metrics	9.39e+03 100 %
monitored_heap_data	6.39e+03 68.1 %
267: heap_data_allocation	6.39e+03 68.1 %
297: monitor_main	6.39e+03 68.1 %
479: main	6.39e+03 68.1 %
2160: operator new[](unsigned long)	7.21e+02 7.7 %
32: operator new(unsigned long)	7.21e+02 7.7 %
52: malloc	7.21e+02 7.7 %
heap_data accesses	7.21e+02 7.7 %

call paths for accesses

```
for( Index_t lnode=0 ; lnode<8 ; ++lnode )
{
    Index_t gnode      = nodelist[ElemId][lnode];
    ElemPos[X_Dir][lnode] = x[gnode];
    ElemPos[Y_Dir][lnode] = y[gnode];
    ElemPos[Z_Dir][lnode] = z[gnode];
}
```

heap data:68%
remote accesses

z accounts for
7.7% remote accesses

remote accesses

z is allocated in a
NUMA domain but
accessed by others

interleave pages of z
across NUMA nodes
13% improvement in
running time



Existing Measurement is Inadequate

- Data collection + attribution $\neq$ optimal optimization
 - know problematic data objects but not know why
 - need more insights for optimization guidance
- Challenges for address sampling
 - very sparse memory access samples
 - not monitoring continuous memory accesses
- Opportunities for address sampling
 - effective addresses: analyze memory access patterns
 - data sources: understand where inefficiencies come from
 - latency: derive new latency metrics to quantify inefficiencies.



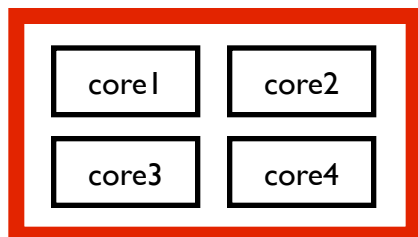
Beyond Data Collection and Attribution

- Published work
 - analyzing NUMA bottlenecks (PPoPP'14)
 - guiding array regrouping for better locality (PACT'14)
 - identifying memory scaling issues (SC'15)

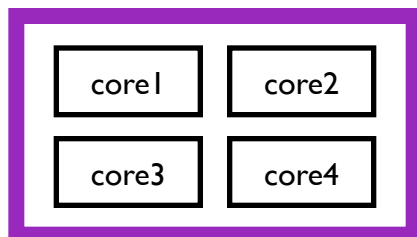


Interleaved Allocation is NOT Always Best

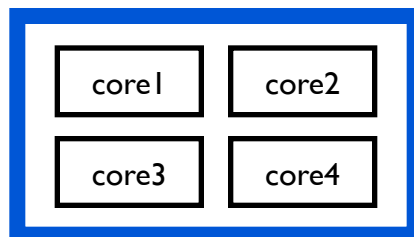
domain 1



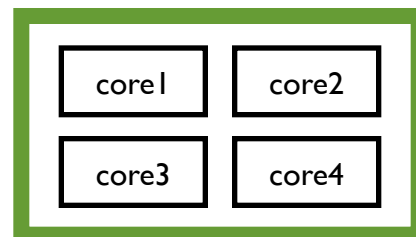
domain 2



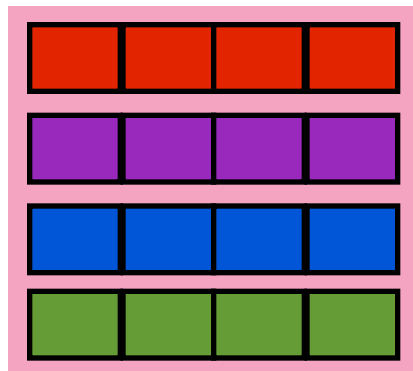
domain 3



domain 4



allocation 1



allocation 2



allocation 3



centralized allocation: poor

interleaved allocation: sub-optimal

co-locate data with computation: optimal

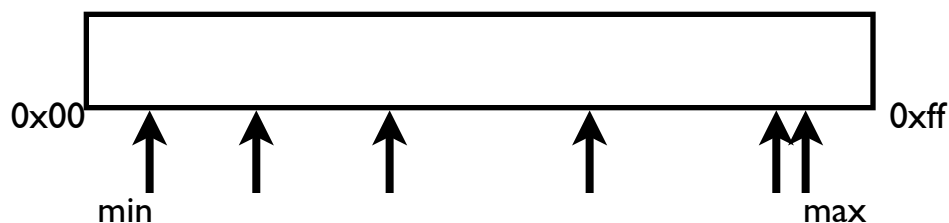
Goal: identify the best data distribution for a program



Memory Access Pattern Analysis

- Online data collection

array A



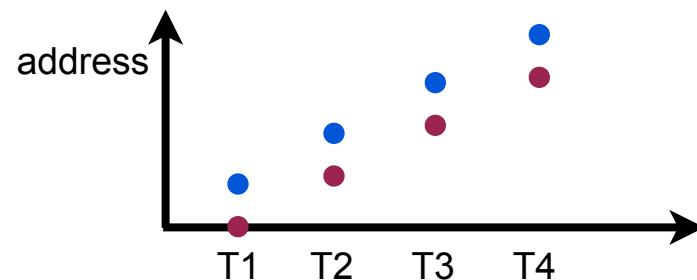
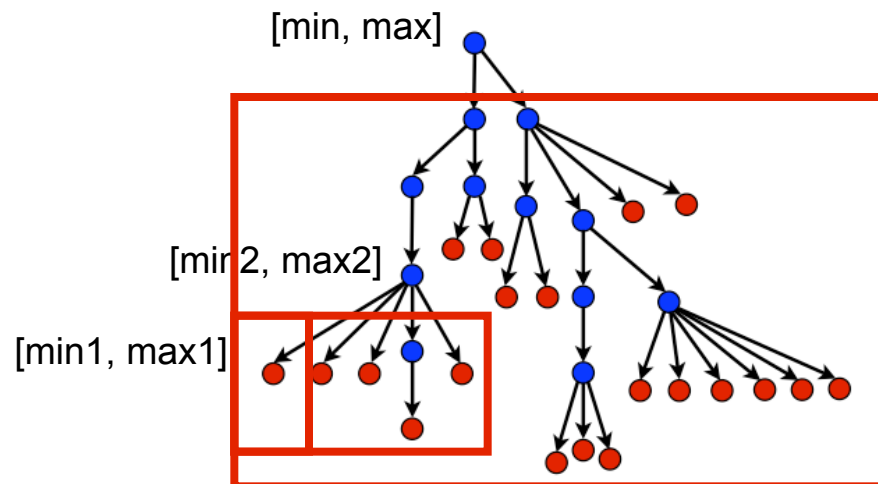
[min, max] per sampled
memory access

balanced allocation + maximum locality

array A

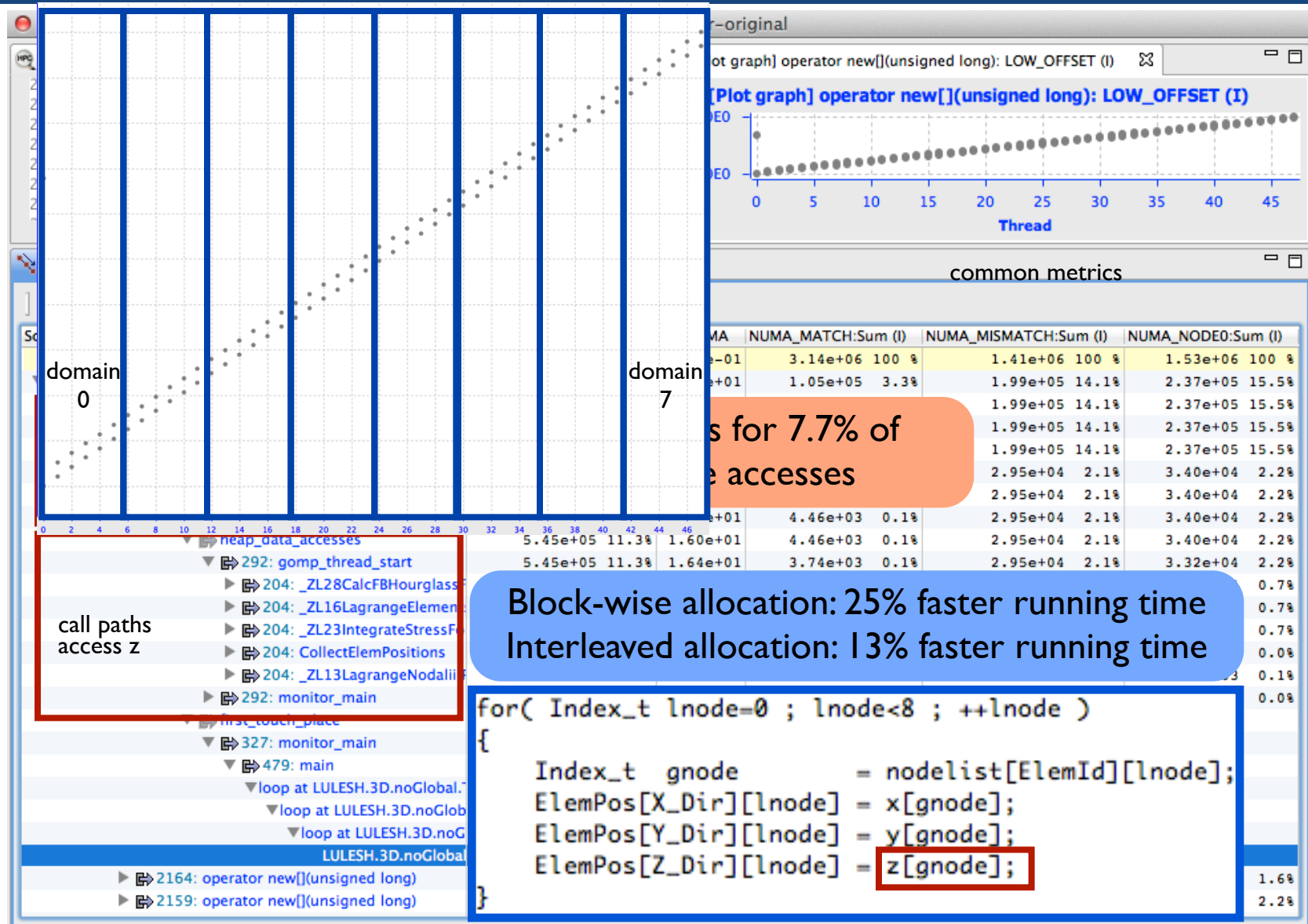


allocate A blockwise to different domains





LULESH on Platform of 8 NUMA Domains





Beyond Data Collection and Attribution

- Published work
 - analyzing NUMA bottlenecks (PPoPP'14)
 - guiding array regrouping for better locality (PACT'14)
 - identifying memory scaling issues (SC'15)

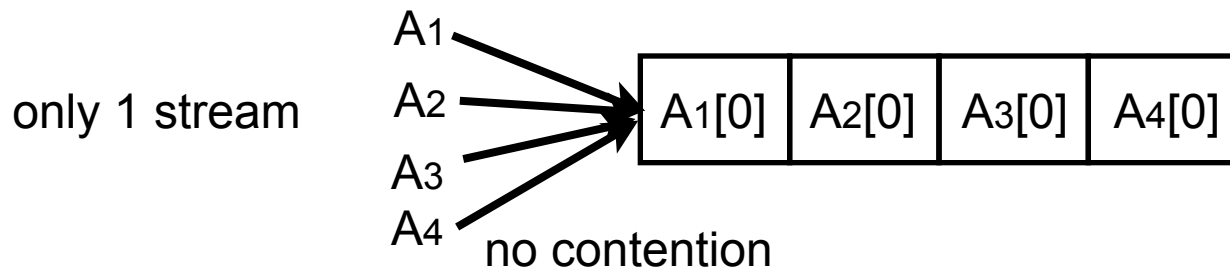
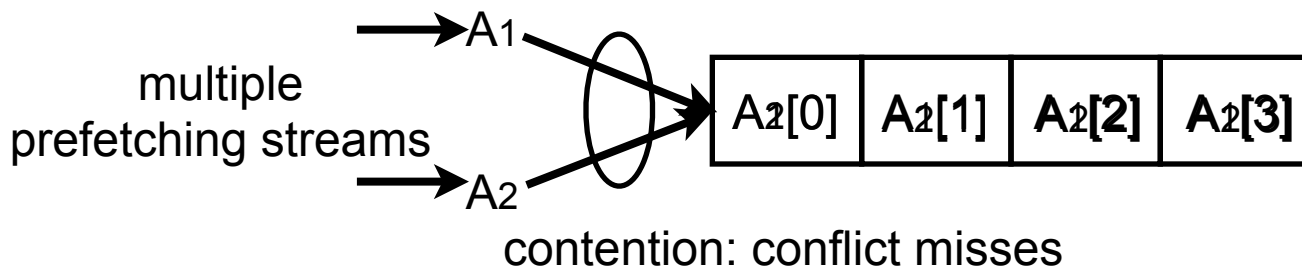


Array Regrouping

```
for (i = 0; i < N; i++)  
  B[i] = A1[i] + A2[i] + ... + An[i];
```

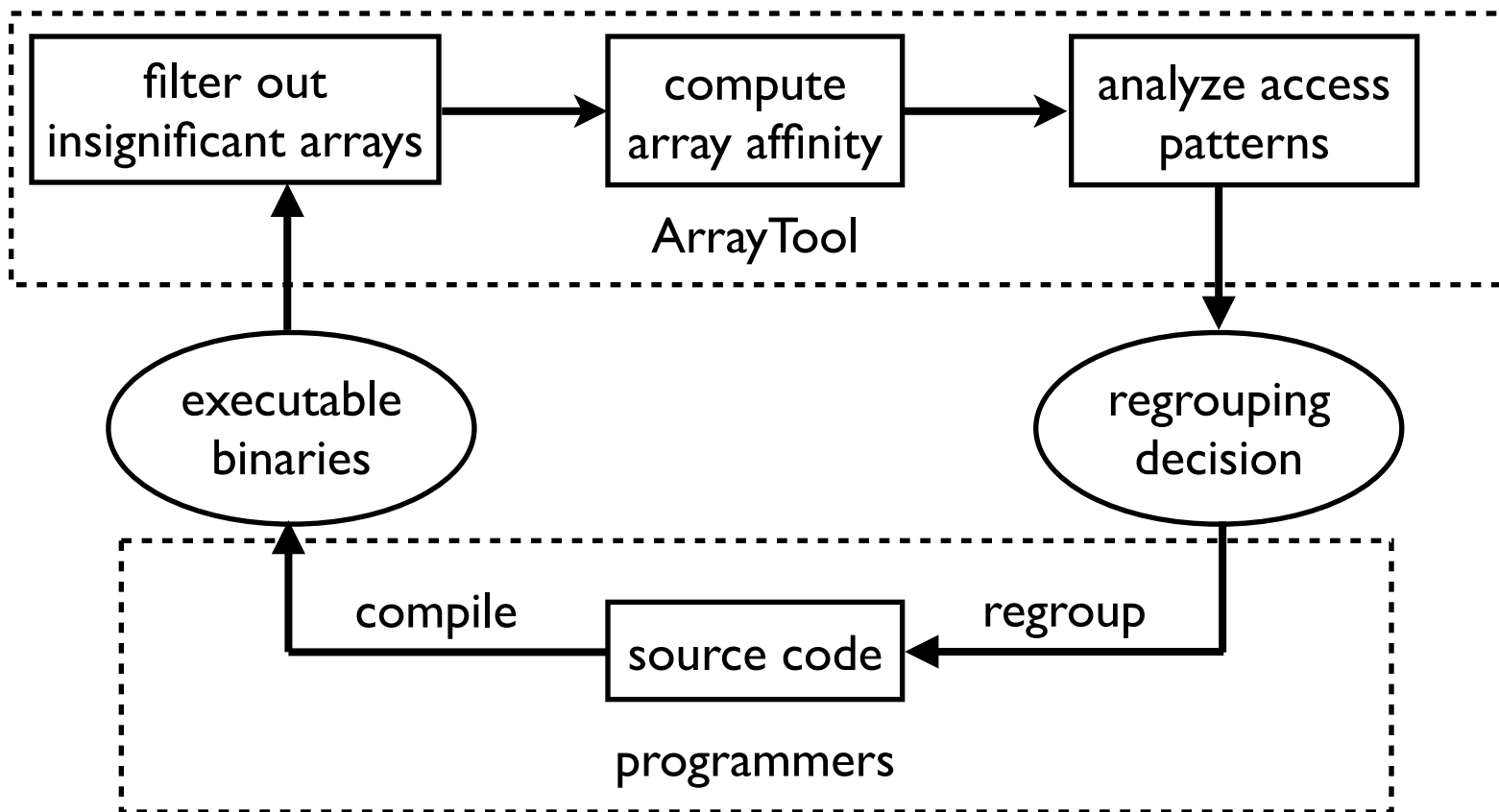


```
for (i = 0; i < N; i++)  
  B[i] = Arr[i].A1 + Arr[i].A2 + ... + Arr[i].An;
```





Workflow





Filter out Insignificant Arrays

latency

```
0.0% I = (float *)malloc( size_I * sizeof(float) );  
12.4% J = (float *)malloc( size_I * sizeof(float) );  
4.9% c = (float *)malloc(sizeof(float)* size_I) ;
```

```
0.0% iN = (int *)malloc(sizeof(unsigned int*) * rows) ;  
0.0% iS = (int *)malloc(sizeof(unsigned int*) * rows) ;  
0.1% jW = (int *)malloc(sizeof(unsigned int*) * cols) ;  
1.5% jE = (int *)malloc(sizeof(unsigned int*) * cols) ;
```

```
2.7% dN = (float *)malloc(sizeof(float)* size_I) ;  
3.8% dS = (float *)malloc(sizeof(float)* size_I) ;  
5.1% dW = (float *)malloc(sizeof(float)* size_I) ;  
5.6% dE = (float *)malloc(sizeof(float)* size_I) ;
```

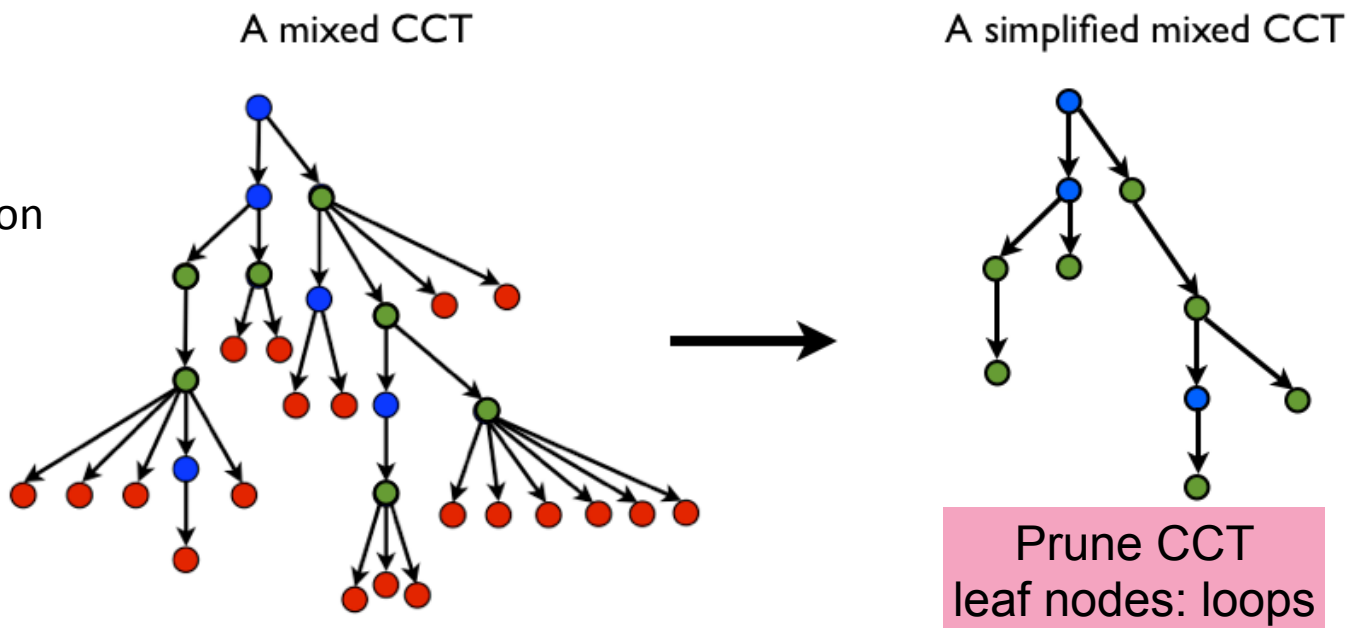
SRAD from Rodinia



Latency-based Array Affinity

loops in their calling contexts

- loop
- instruction
- function



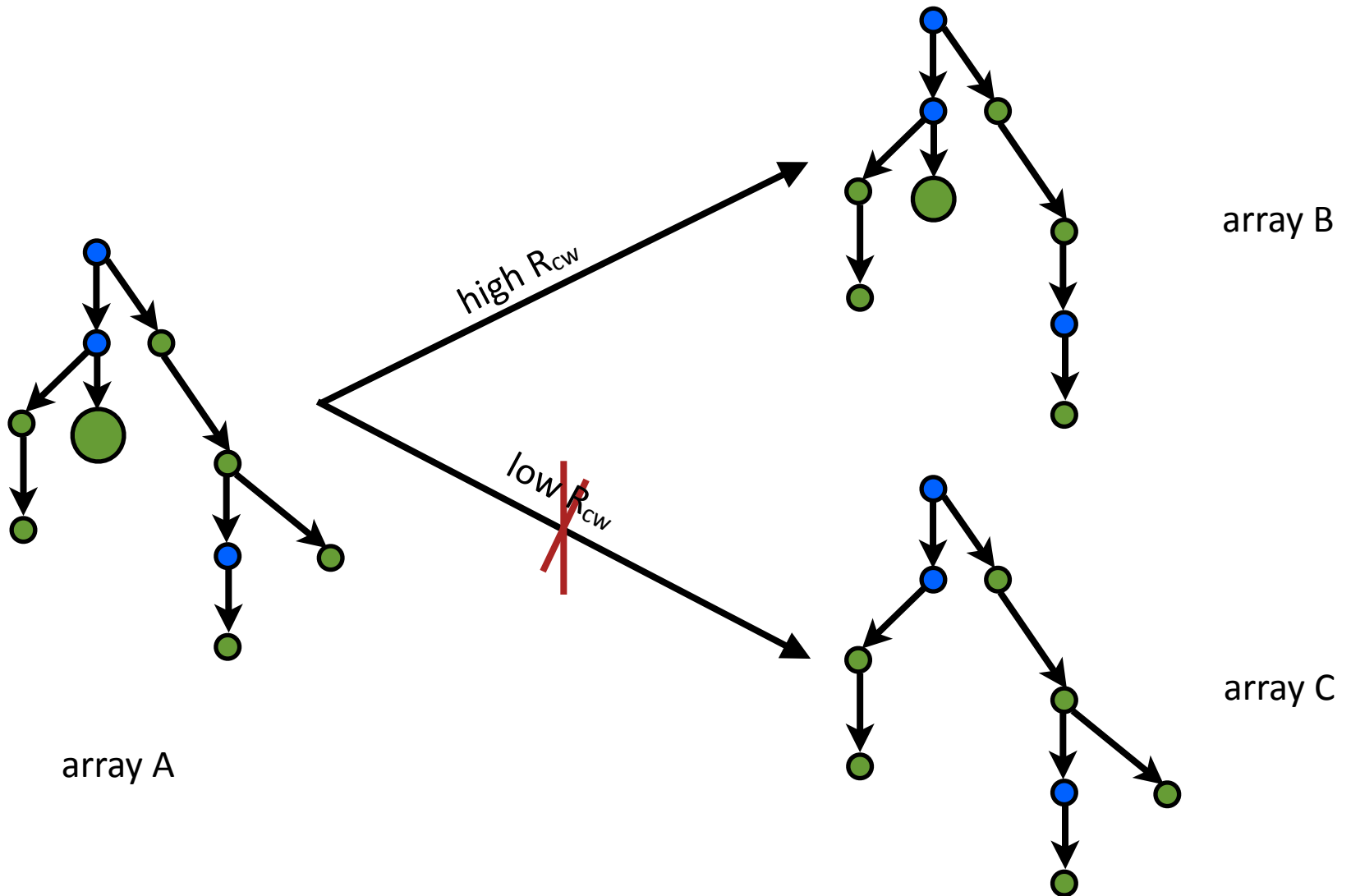
$$R_{cw} = 1 - \frac{W_{uncommon}}{W_{total}}$$

aggregate latency of uncommon loops

aggregate latency of all loops

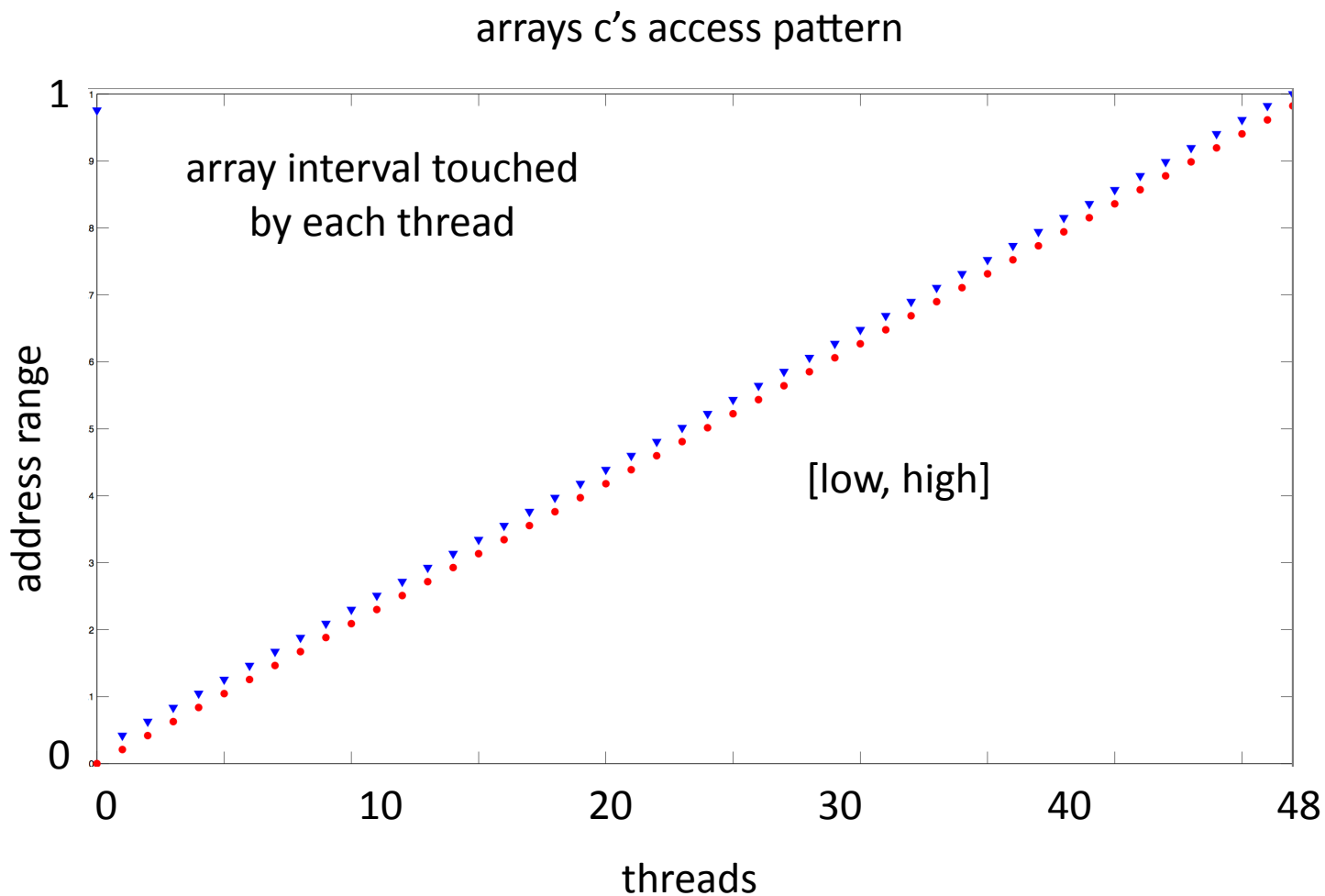


An Example





Access Pattern Analysis



dN, dS, dE, dW have the same pattern



Regrouping Results of SRAD

```
#pragma omp parallel for
for(int i=0; i<rows; i++) {
    for(int j=0; j<cols; j++) {
        k = i*cols+j;
        cN = c[k];
        cS = c[iS[i]*cols+j];
        cW = c[k];
        cE = c[i*cols+jE[j]];
        D = cN*dN[k] + cS*dS[k] + cW*dW[k] + cE*dE[k];
        J[k] = J[k] + 0.25*lambda*D;
    }
}
```

only regroup dN, dS, dW, dE $\longrightarrow$ $\uparrow 1.47x$

regroup dN, dS, dW, dE and c $\longrightarrow$ $\uparrow 1.89x$



Beyond Data Collection and Attribution

- Published work
 - analyzing NUMA bottlenecks (PPoPP'14)
 - guiding array regrouping for better locality (PACT'14)
 - identifying memory scaling issues (SC'15)



Scaling Losses in Memory Hierarchies

- Memory contentions hurt scalability: cache/bandwidth contention
 - which data objects contribute to the most scaling losses
 - which memory layers incur the most scaling losses
- Methods
 - decompose latency according to data objects and memory layers
 - data-centric analysis with data source information
 - differential analysis supported by HPCToolkit
 - compare profiles between different runs

more details in SC'15



Conclusions and Future Work

- Hardware address sampling
 - widely supported in modern architectures
 - powerful in monitoring memory behaviors
 - more analysis of the samples provides more performance insights
- On-going work
 - structure splitting
 - locality optimization between SMT threads
 - cache line false sharing
 - automatic page migration for NUMA architectures
- Future directions of address sampling
 - comparing different address sampling mechanisms
 - analyzing new performance issues
 - heterogeneous memory: 3D stack memory





Backup Slides



UMT2013 on Quad-socket POWER7 Node

ZoneData_mod.F90 [Plot graph] malloc: LOW_OFFSET (l)

```
90 allocate( self % A_fp(Size% ndim,self% nCFaces,self% nCorner) )
91 allocate( self % A_ez(Size% ndim,self% nCFaces,self% nCorner) )
92 allocate( self % Connect(3,self% nCFaces,self% nCorner) )
93 allocate( self % STotal(Size% ngr,self% nCorner) )
94 allocate( self % STime(Size% ngr,self% nCorner,Size% nangSN) )
```

Calling Context View

Scope

Scope	NUMA_MISMATCH:Sum (l)
Experiment Aggregate Metrics	2.45e+04 100 %
▶ monitored_unknown_data	1.30e+04 53.1 %
▼ monitored_heap_data	1.15e+04 46.9 %
▼ 266: heap_data_allocation	1.15e+04 46.9 %
▼ 296: monitor_main	1.10e+04 44.9 %
▼ 479: main	1.10e+04 44.9 %
▼ inlined from SuOlsonTest.cc: 67	1.10e+04 44.8 %
▼ 167: initialize(Geometry::MeshBase&, Teton<Geometry::MeshBase>&)	1.05e+04 42.9 %
▼ 314: Teton<Geometry::MeshBase>::linkKull(Geometry::MeshBase&)	1.05e+04 42.9 %
▼ loop at Teton.cc: 1250	4.45e+03 18.2 %
▼ 1328: setzone	4.45e+03 18.2 %
▼ 40: zonedata_ctor	4.45e+03 18.2 %
▼ loop at ZoneData_mod.F90: 86	4.45e+03 18.2 %
▼ loop at ZoneData_mod.F90: 87	4.45e+03 18.2 %
▼ loop at ZoneData_mod.F90: 88	4.45e+03 18.2 %
▼ loop at ZoneData_mod.F90: 89	4.45e+03 18.2 %
▼ loop at ZoneData_mod.F90: 90	4.45e+03 18.2 %
▼ loop at ZoneData_mod.F90: 91	4.45e+03 18.2 %
▼ loop at ZoneData_mod.F90: 92	4.45e+03 18.2 %
▼ loop at ZoneData_mod.F90: 93	4.45e+03 18.2 %
▼ loop at ZoneData_mod.F90: 94	4.45e+03 18.2 %
▼ 94: malloc	4.45e+03 18.2 %
▼ heap_data_accesses	4.45e+03 18.2 %
▼ 291: ThdCode	4.45e+03 18.2 %
▼ _xlsmpl_DynamicCh	4.45e+03 18.2 %
▼ snflwxyz\$\$OL55	4.45e+03 18.2 %

sample off-chip
accesses

self%STime

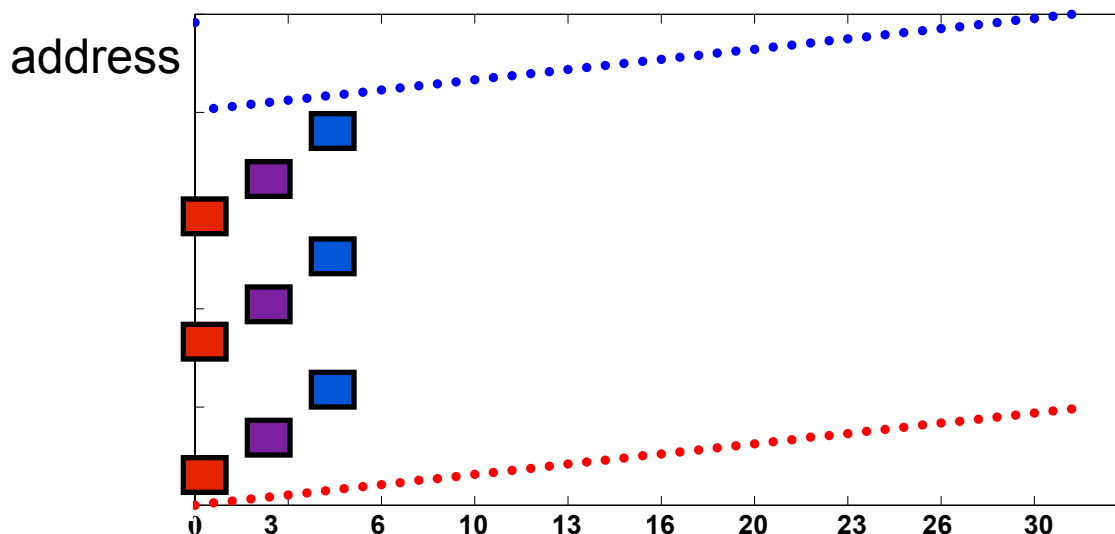
18.2% of remote accesses

allocated in one domain
accessed by everyone



Optimize *self%STime* for UMT2013

address-centric analysis for *self%STime*



self%STime's address space



optimization: let each thread
initialize its own data

result: all threads have data
locally -- 7% faster