

---

# Leveraging Hardware Address Sampling for Memory Performance Insights

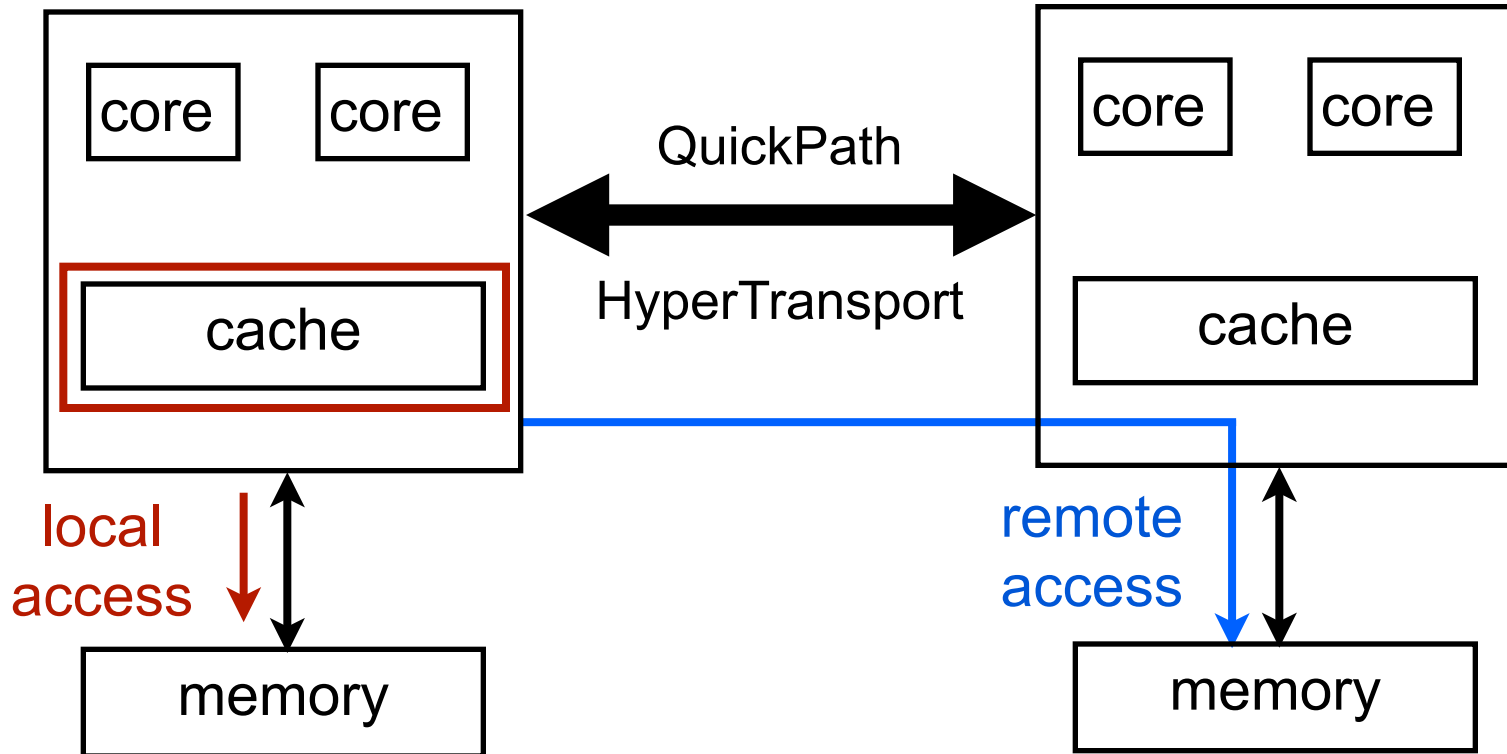
Xu Liu

Department of Computer Science  
College of William and Mary  
[xl10@cs.wm.edu](mailto:xl10@cs.wm.edu)



# Motivation: Memory is the Bottleneck

NUMA: Non-Uniform Memory Access





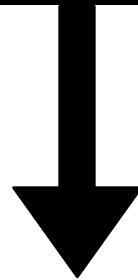
# State of the Arts

simulation methods

deep insights

weaknesses:

- 2-5x overhead
- not real machines

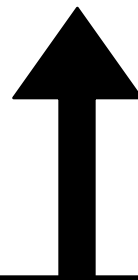


low overhead with deep insights

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

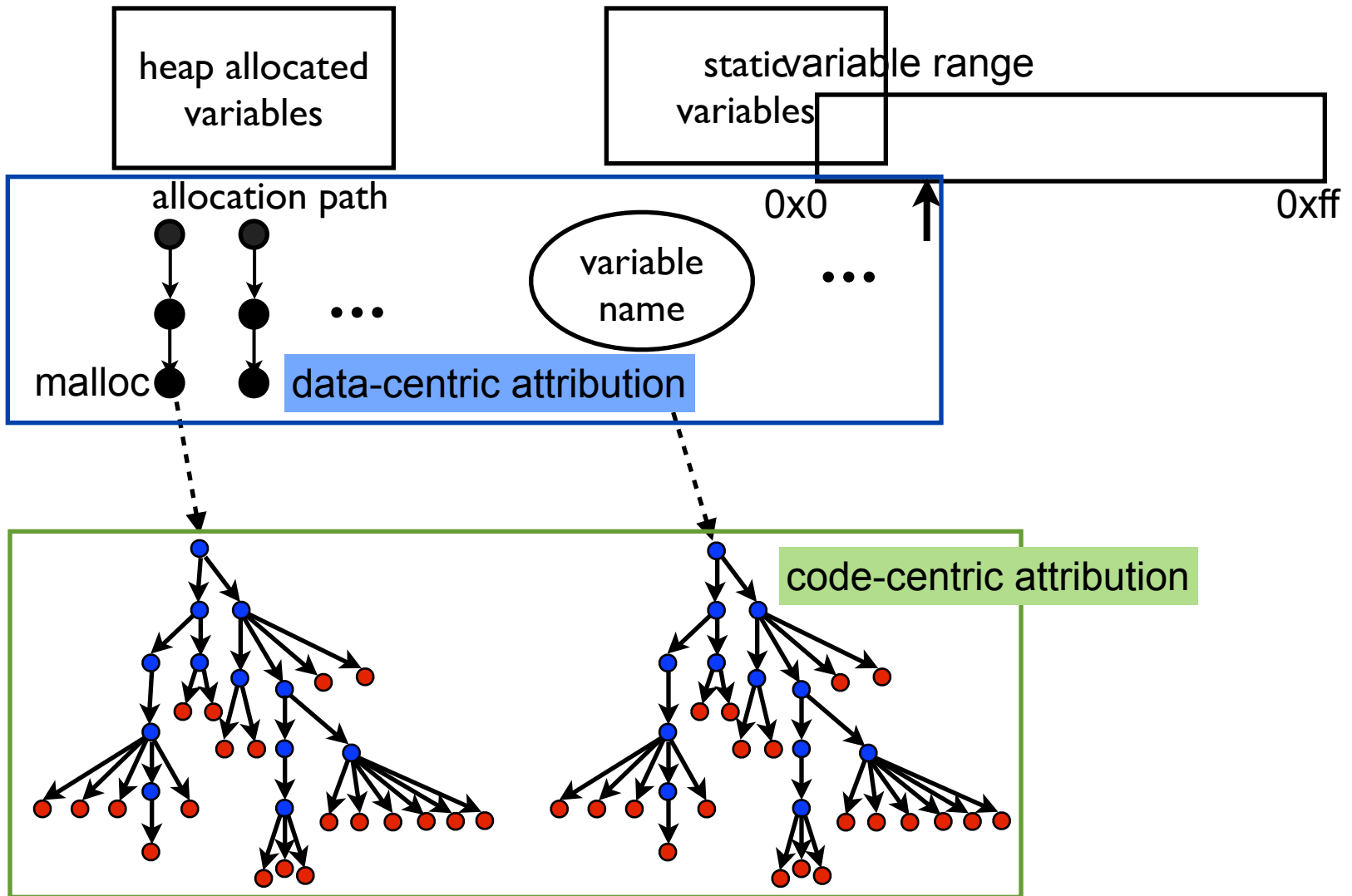
array C

line 5: 89% latency

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

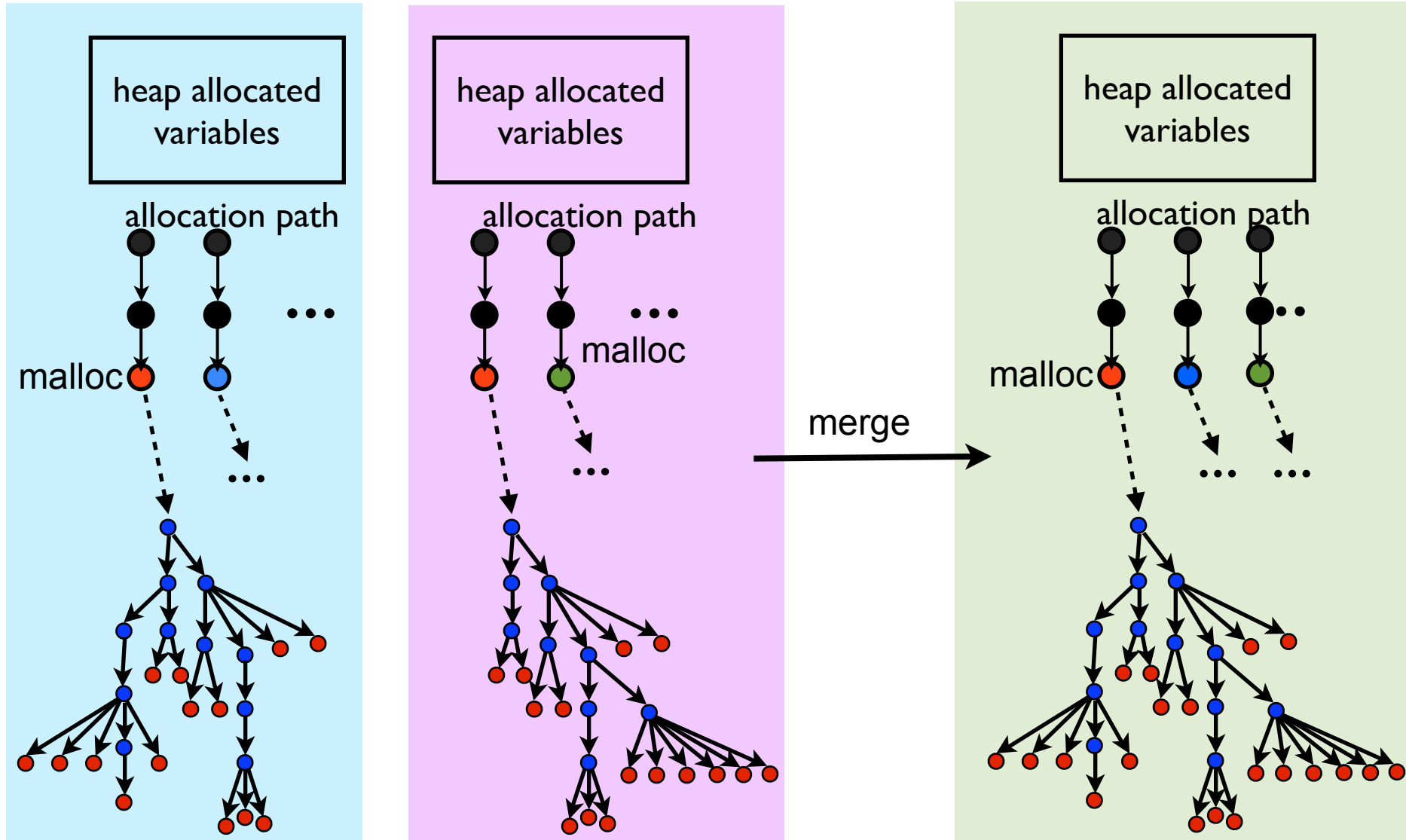


# Attributing Samples





# Aggregating Profiles





# LULESH on Platform of 8 NUMA Domains

The screenshot shows a debugger window with the following components:

- Source Code:** Lines 2158-2161 show the declaration of arrays `*x`, `*y`, `*z`, and `*xd`. Line 2160 is highlighted in blue.
- Calling Context View:** A tree view showing the call path from `main` down to `52: malloc`. A blue arrow labeled "allocation call path" points to `52: malloc`. A blue box labeled "call site of allocation" is also around `52: malloc`.
- Performance Table:** A table with columns "Scope" and "R\_DRAM\_ACCESS:Sum (l)".
- Code Snippet:** A `for` loop with `z[gnode]` circled in red. A green box labeled "call paths for accesses" points to this line.

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%

```
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.



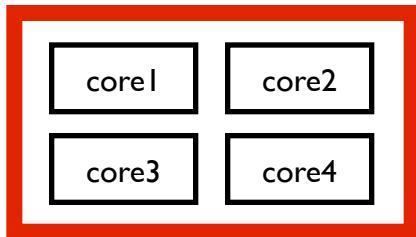
# Beyond Data Collection and Attribution

- Published work
  - HPCToolkit-NUMA: analyzing NUMA bottlenecks (PPoPP'14)
  - ArrayTool: guiding array regrouping for better locality (PACT'14)
  - ScaAnalyzer: identifying memory scaling issues (SC'15)
  - StructSlim: guiding structure splitting (CGO'16)
  - Cheetah: detecting false sharing (CGO'16)
  - SMTAnalyzer: identifying SMT-aware optimization (HPDC'16)

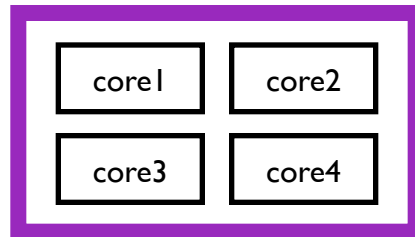


# Interleaved Allocation is NOT Always Best

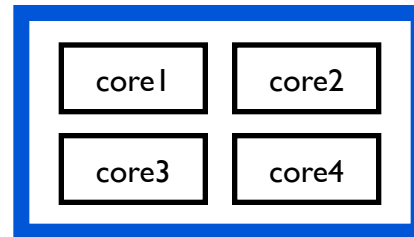
domain 1



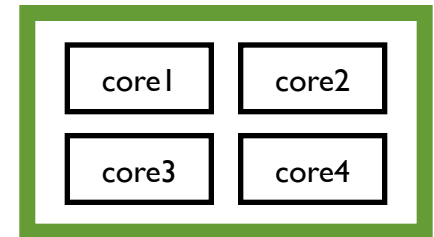
domain 2



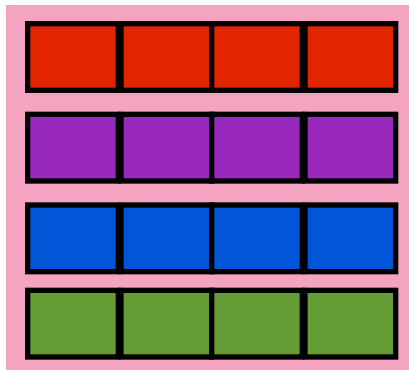
domain 3



domain 4



allocation 1



allocation 2



allocation 3



centralized allocation: poor

interleaved allocation: sub-optimal

co-locate data with computation: optimal

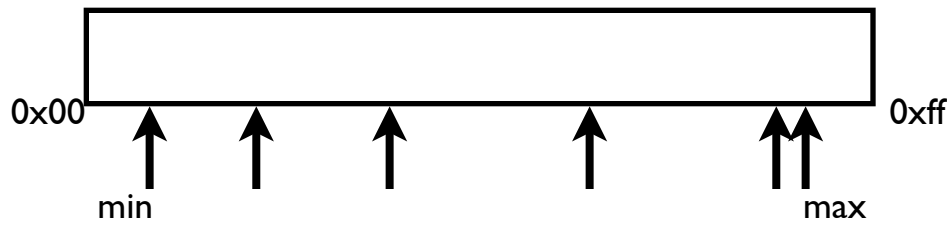
Goal: identify the best data distribution for a program



# Memory Access Pattern Analysis

- Online data collection

array A



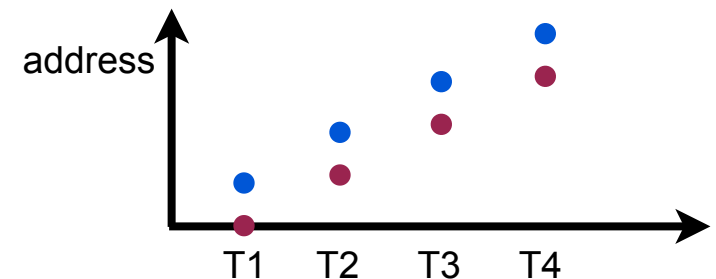
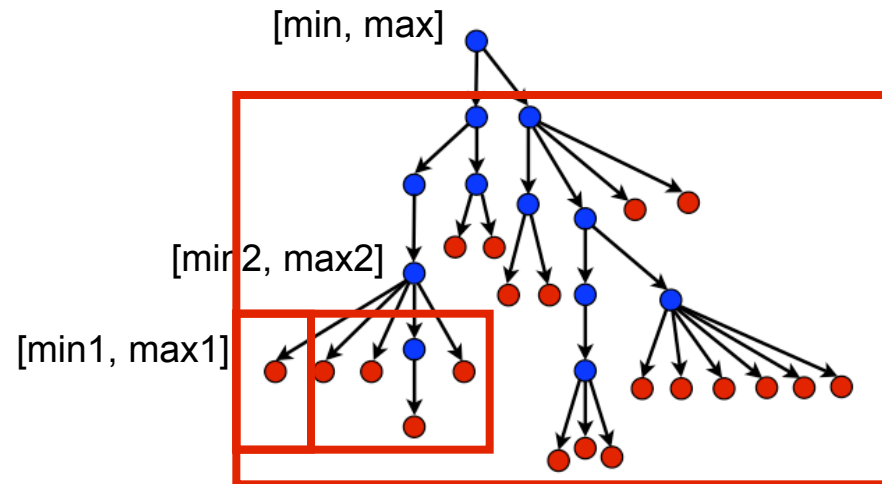
[min, max] per sampled memory access

balanced allocation + maximum locality

array A

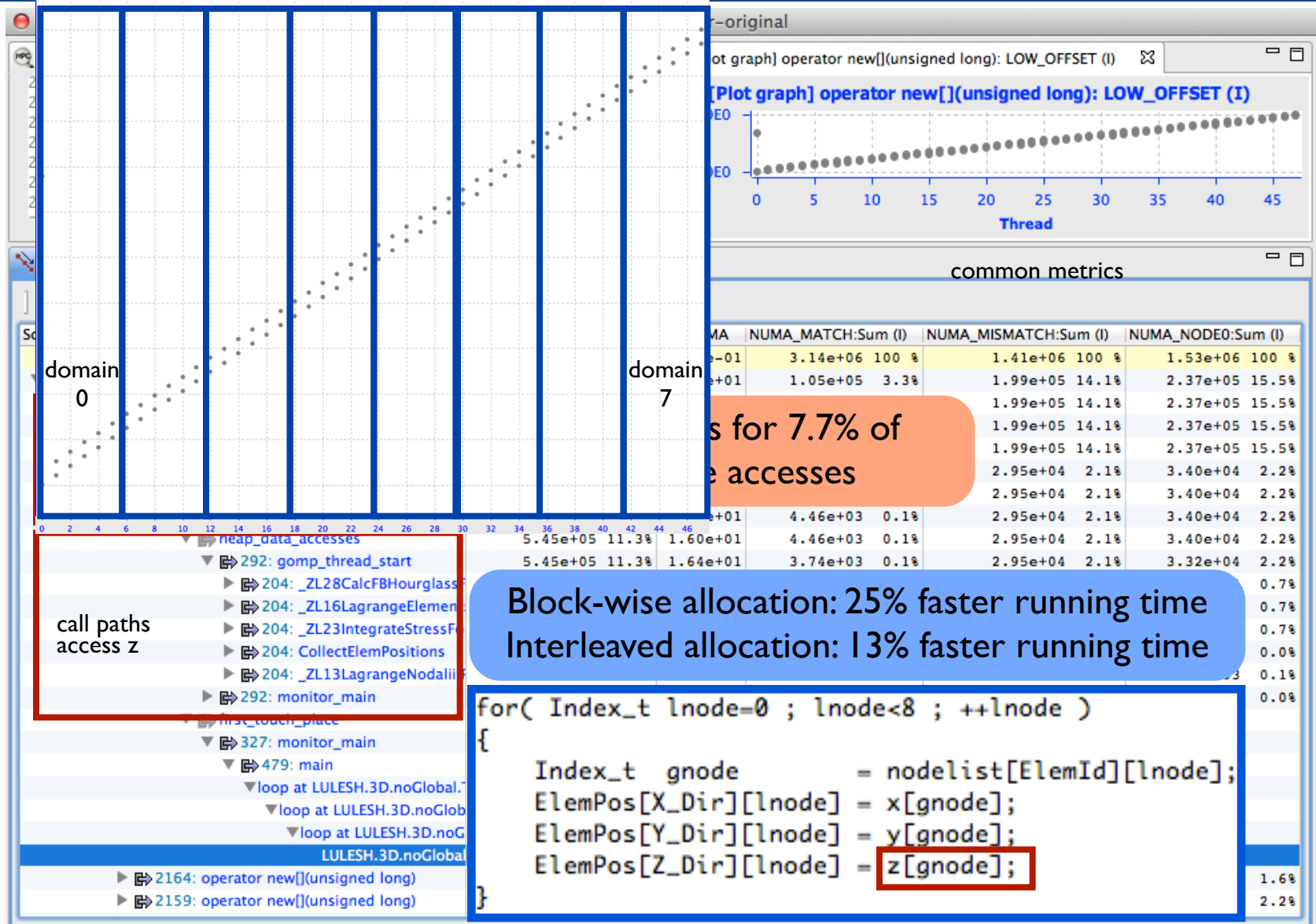


allocate A blockwise to different domains





# LULESH on Platform of 8 NUMA Domains



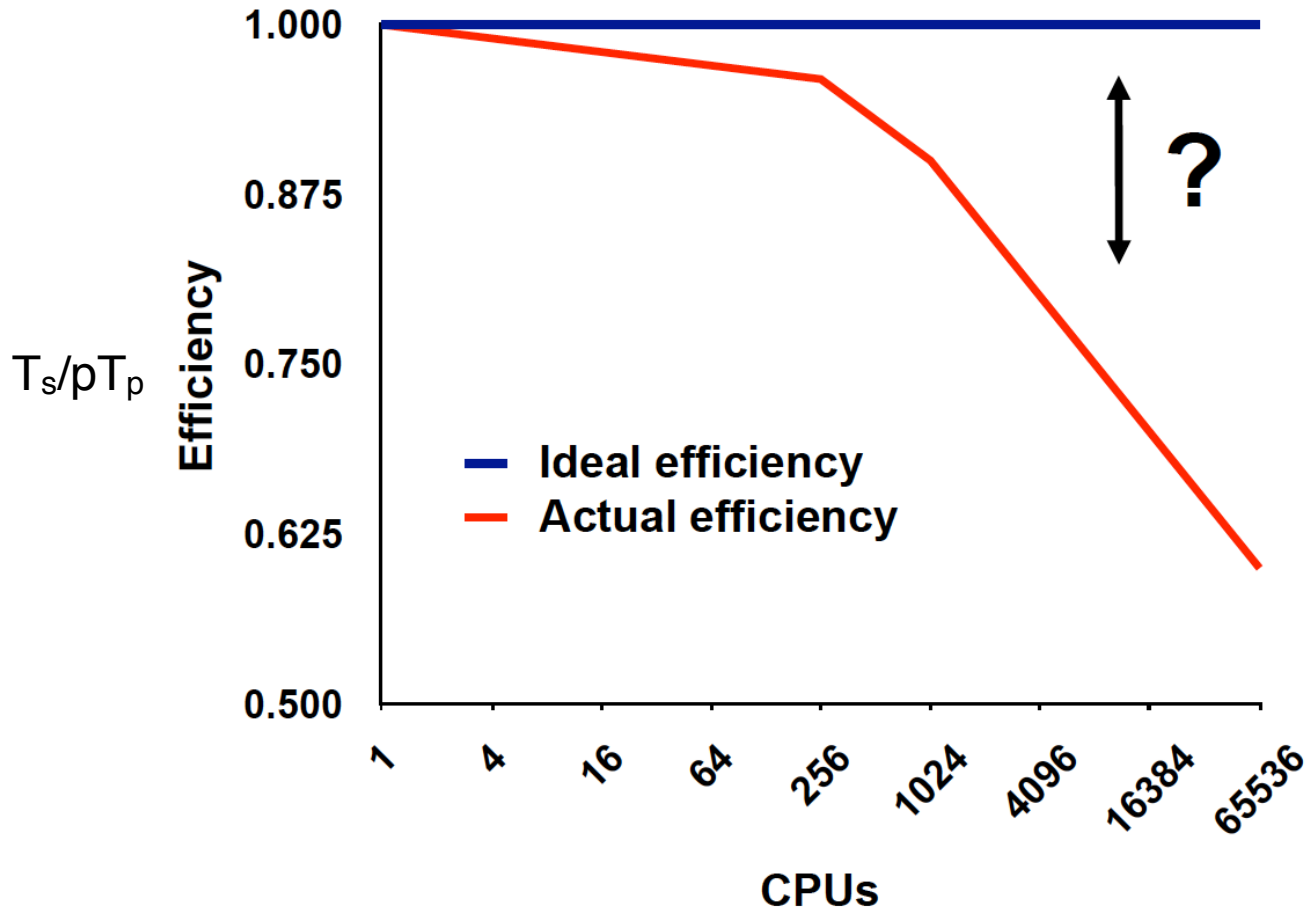


# Beyond Data Collection and Attribution

- Published work
  - HPCToolkit-NUMA: analyzing NUMA bottlenecks (PPoPP'14)
  - ArrayTool: guiding array regrouping for better locality (PACT'14)
  - ScaAnalyzer: identifying memory scaling issues (SC'15)
  - StructSlim: guiding structure splitting (CGO'16)
  - Cheetah: detecting false sharing (CGO'16)
  - SMTAnalyzer: identifying SMT-aware optimization (HPDC'16)



# The Problem of Scaling



Note: higher is better



# Quantifying Scaling Loss

- Quantifying scaling loss with execution time (parallel efficiency)
  - it is straightforward with expectation: the same program with the same workloads

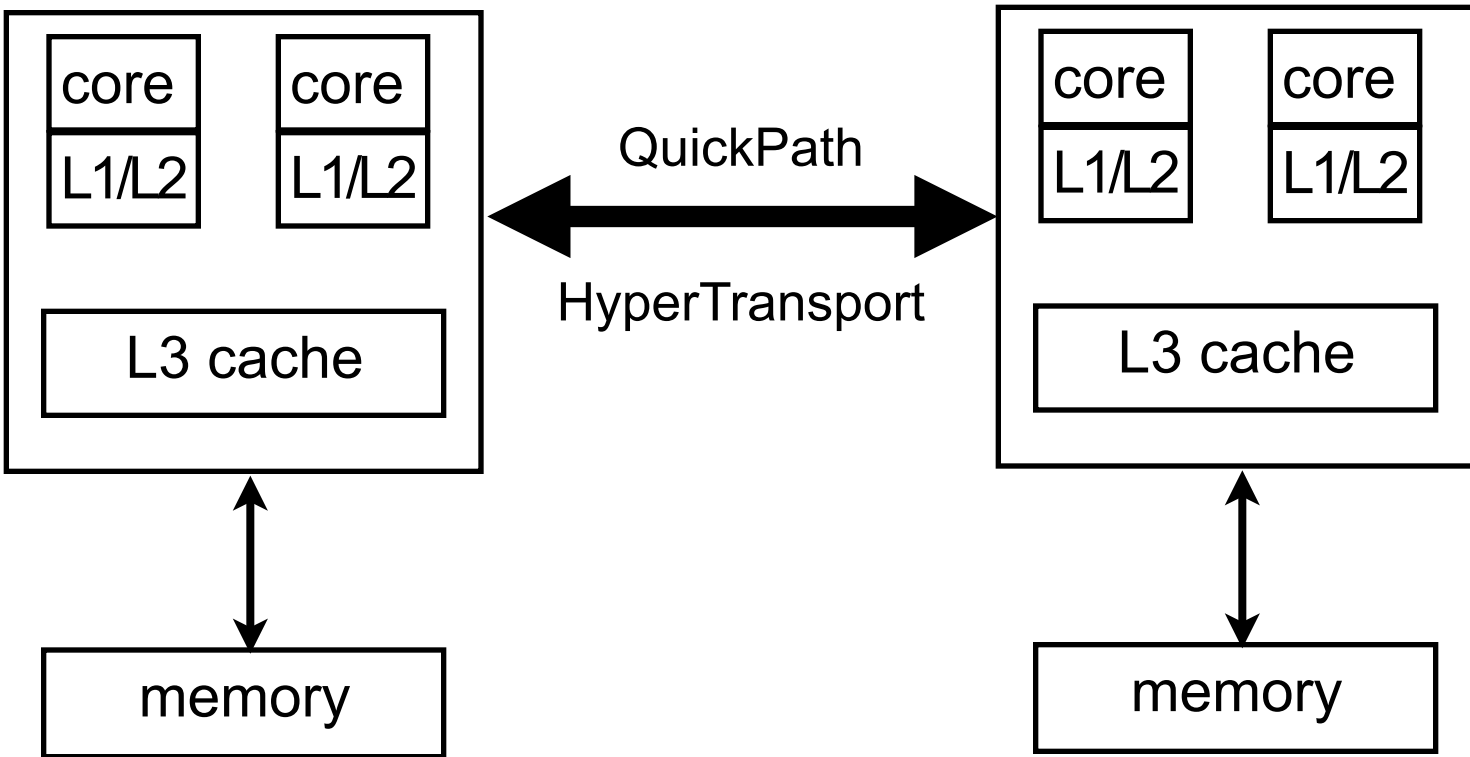
$$T_{2n} = T_n/2$$

- $T_{2n} = T_n/2$ : perfect scaling — expected
  - $T_{2n} > T_n/2$ : scaling with loss
  - $T_{2n} < T_n/2$ : superlinear scaling
- execution time cannot help us focus on memory scaling
    - can we have an expectation for the memory performance for scaling?





# Memory Scaling Expectation



The average latency of memory accesses should be at least the same with perfect scaling



# Derived Metrics

- Average latency  $l_p$  over all memory accesses on  $p$  cores  
expectation: **scaling factor** =  $l_p/l_q = 1$  ( $p > q$ )
  - if  $l_p/l_q \gg 1$ 
    - there is significant scaling loss in memory
  - which memory layer has the most memory scaling loss
  - which part of the source code has the memory scaling loss



# Root Cause Analysis for Memory Scaling

- Which memory layer causes the scaling loss?
  - private layers: L1, L2 caches
  - shared layers: L3 cache and main memory
  - NUMA layers: remote L3 cache and remote memory

Provide optimization guidance: eliminating false sharing, mitigating contention, or addressing NUMA bottlenecks

- Which data objects and memory accesses cause the scaling loss?
  - problematic arrays with problematic accesses

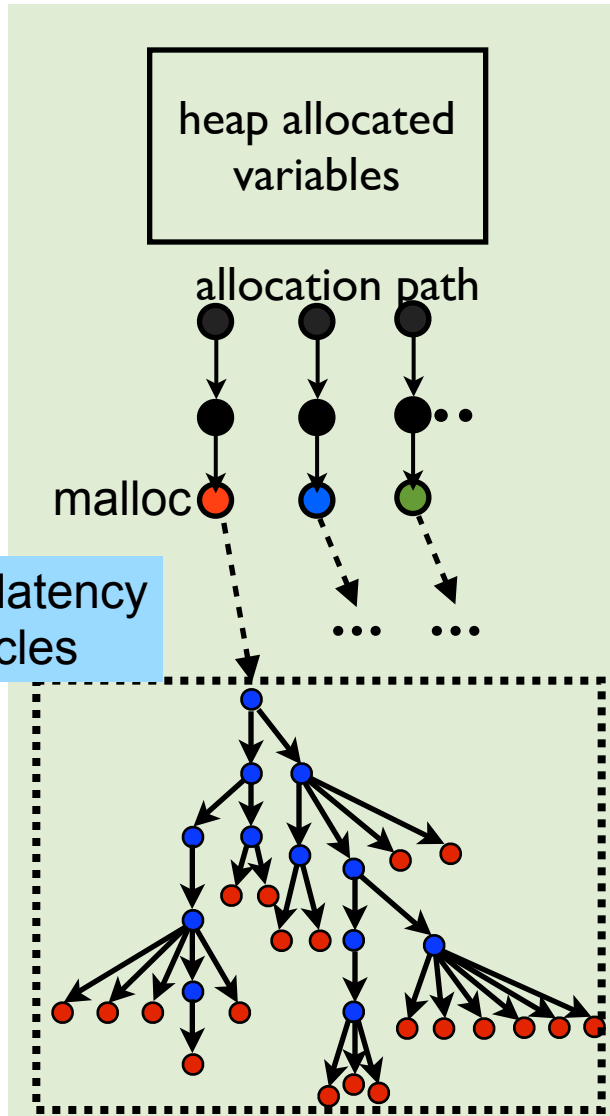
Pinpointing problematic source code for optimization: high-level feedback for programmers

ScaAnalyzer extends HPCToolkit:  
lightweight analysis

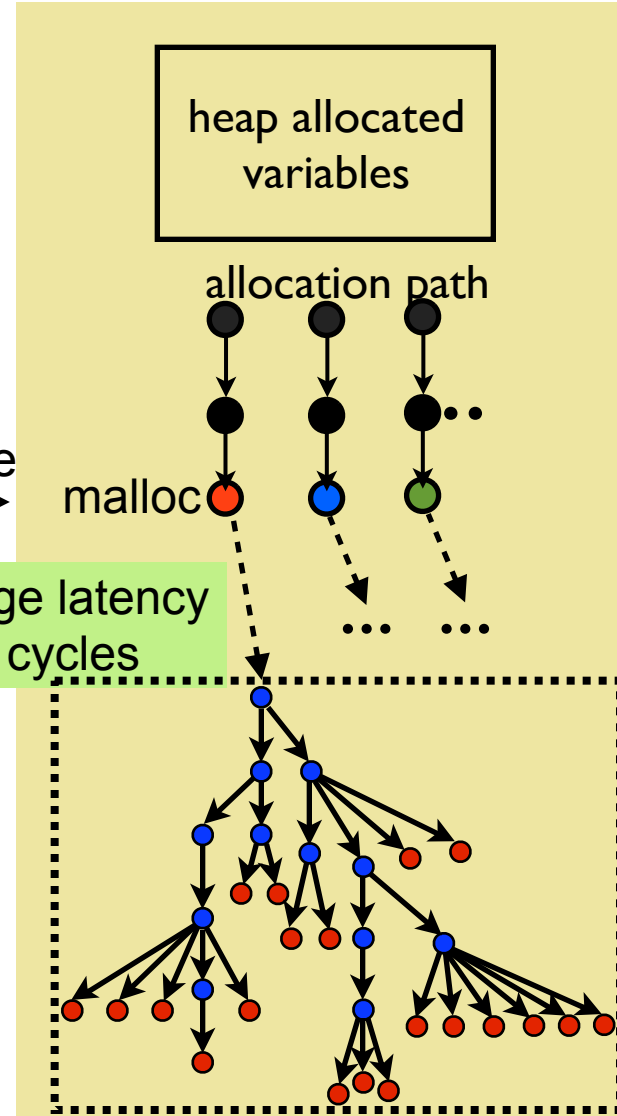


# Differential Analysis on Aggregate Profiles

N cores



2N cores





# IRSmk: An Important DOE Benchmark

```
aos3.cpp 309
310     int size = i_ub;
311
312     double *b = new double[i_ub];
313     double *x = new double[x_size];
314     my_type *ABCD = new my_type[size];
315
316     /*if ((ufr) == NULL)
317         print("*****ERROR: allocMem out of memory \n");*/
```

Calling Context View | Callers View | Flat View

16 cores | 32 cores

Scope	1.LATENCY:Sum (l)	2.LATENCY:Sum (l)	scaling factor	NUMA scaling factor
Experiment Aggregate Metrics	6.55e+06 100 %	1.01e+07 100 %	2.02e+00	2.54e+00
▼ monitored_heap_data	6.50e+06 99.2%	9.93e+06 98.8%	2.04e+00	2.26e+00
▼ heap_data_allocation	6.50e+06 99.2%	9.93e+06 98.8%	2.04e+00	2.26e+00
▼ monitor_main	6.50e+06 99.2%	9.93e+06 98.8%	2.04e+00	2.26e+00
▼ main	6.50e+06 99.2%	9.93e+06 98.8%	2.04e+00	2.26e+00
▼ operator new[...]	6.38e+06 97.4%	9.72e+06 96.7%	2.03e+00	2.19e+00

allocation call path

NUMA layer impedes the scalability



# Conclusions and Future Work

- Hardware address sampling
  - widely supported in modern architectures
  - powerful in monitoring memory behaviors
  - further analysis of the samples provides deeper performance insights
- On-going work
  - automatic page migration for NUMA architectures
  - program optimization for heterogenous memories
    - with memif support (goto memif talk at ASPLOS)
- Future work: profilers for Big Data workloads
  - understand memory contention, memory usage, and SMT effects
  - multiple programs co-running
  - go beyond code-centric and data-centric analysis





# Backup Slides

---



# UMT2013 on Quad-socket POWER7 Node

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

```
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	NUMA_MISMATCH:Sum (I)
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

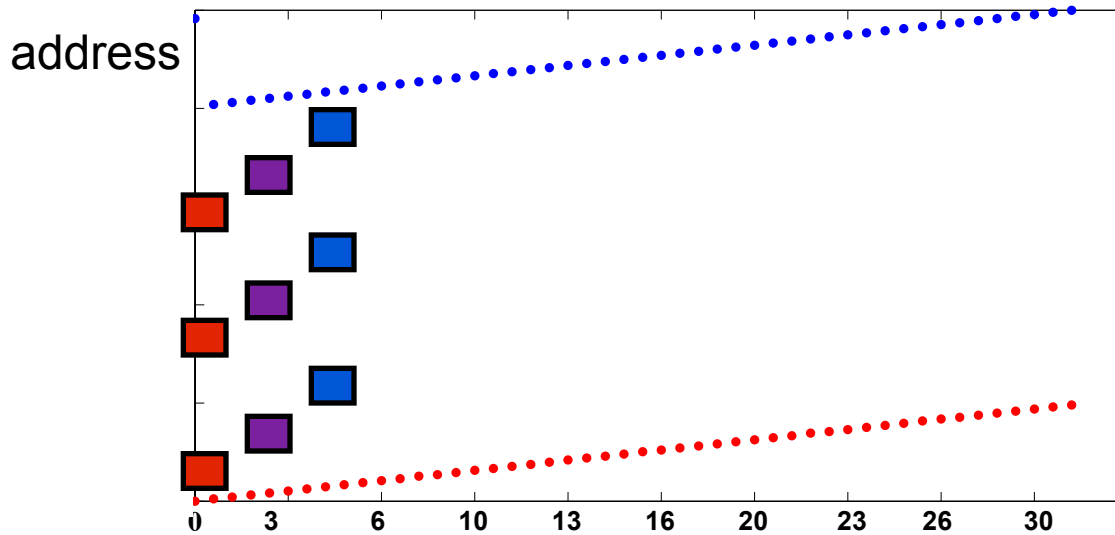
self%STime  
18.2% of remote accesses  
allocated in one domain  
accessed by everyone





# 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