Purity: An Integrated, Fine-Grain, Data-Centric, Communication Profiler for the Chapel Language

Richard B. Johnson and Jeffrey K. Hollingsworth
Department of Computer Science, University of Maryland, College Park

CHIUW
05/25/2018

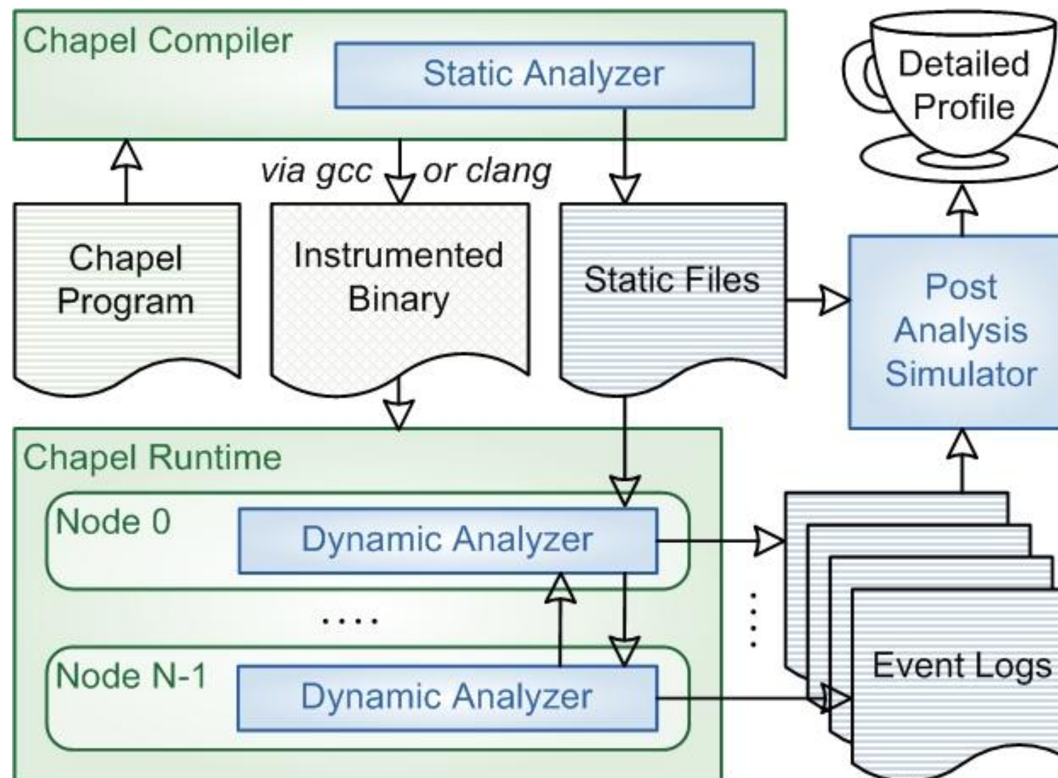
Motivation and Concept

- Our goals for developing a profiling system
  - Analyze memory and communication access patterns
  - Support for multi-node PGAS environments
  - Integrate profiler into the Chapel framework
- General approach
  1. Identify and instrument remotely accessible variables in user modules through the compiler
  2. Map the memory addresses referenced by remote operations at runtime to variable definitions
  3. Perform an analysis over the runtime execution

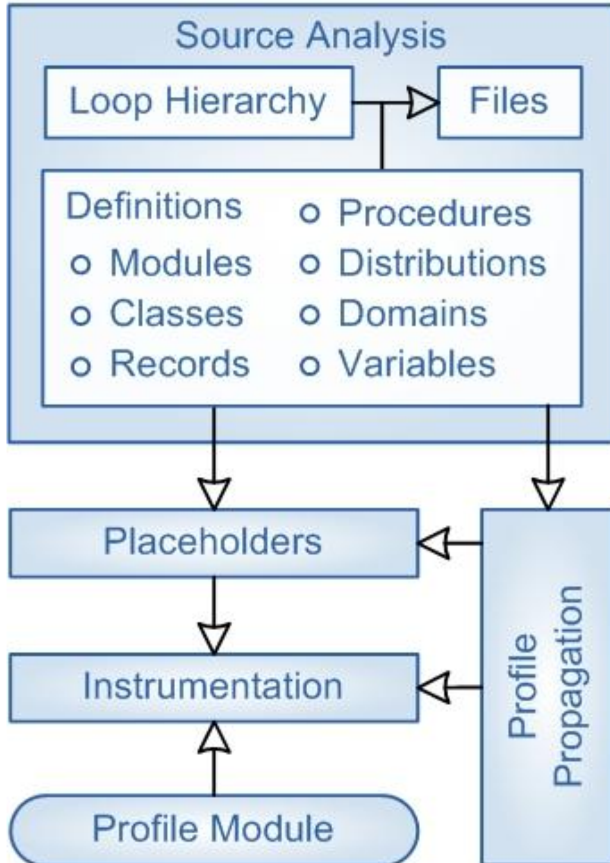
Challenge and Direction

1. Unable to identify remote accesses until late pass
   - Placeholders and late stage instrumentation are required
2. Each node only has a partial view of the PGAS
   - Unable to map remote addresses to variable definitions
   - Solution: Resolve addresses in a post analysis setting
     - Unified view of PGAS in an offline analysis
     - Communication / memory operations need to be recorded
     - The memory states of the nodes need to be synchronized

Design Overview

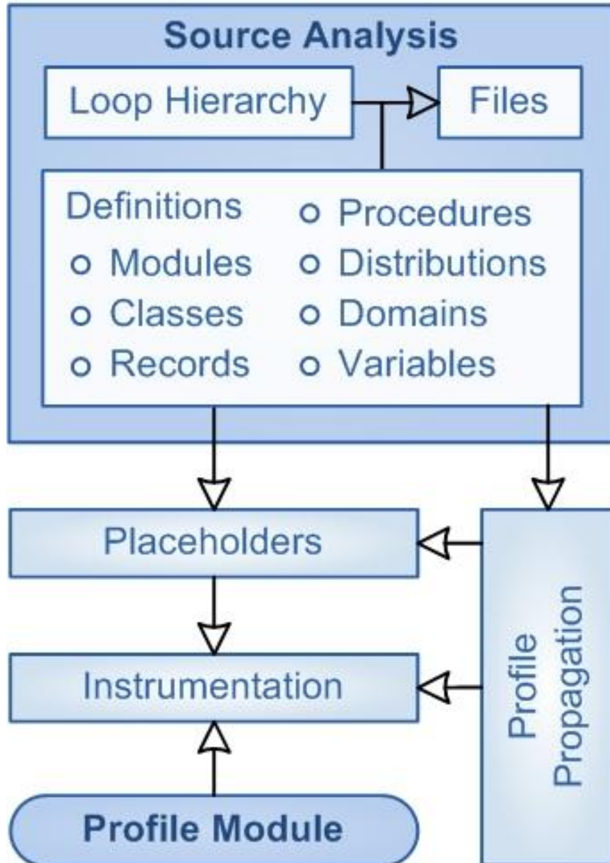


Static Analyzer



- Adds five new passes to compiler

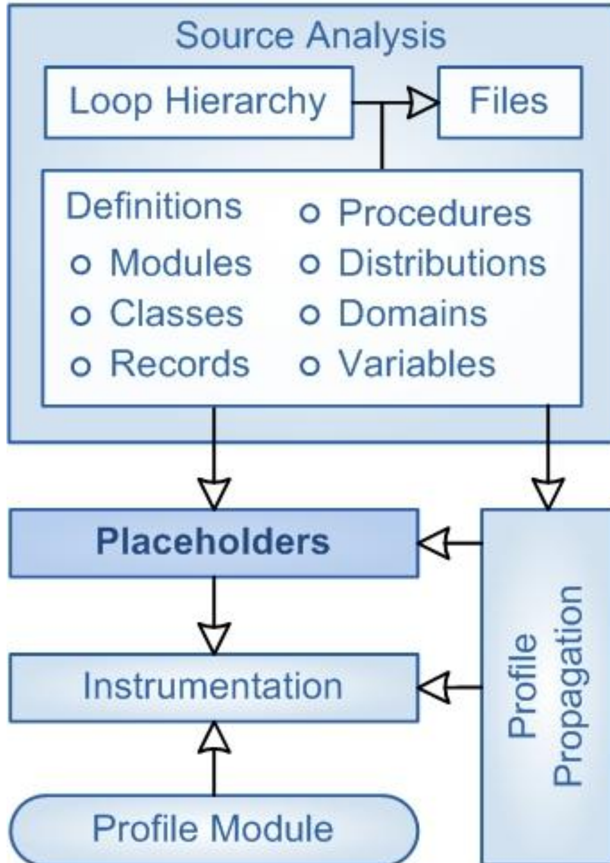
Static Analyzer



- Adds five new passes to compiler

1. Source analysis pass
   - Loads profile module and profile input file
   - Analyzes program ASTs after parser
   - Acquires and stores maps and definitions

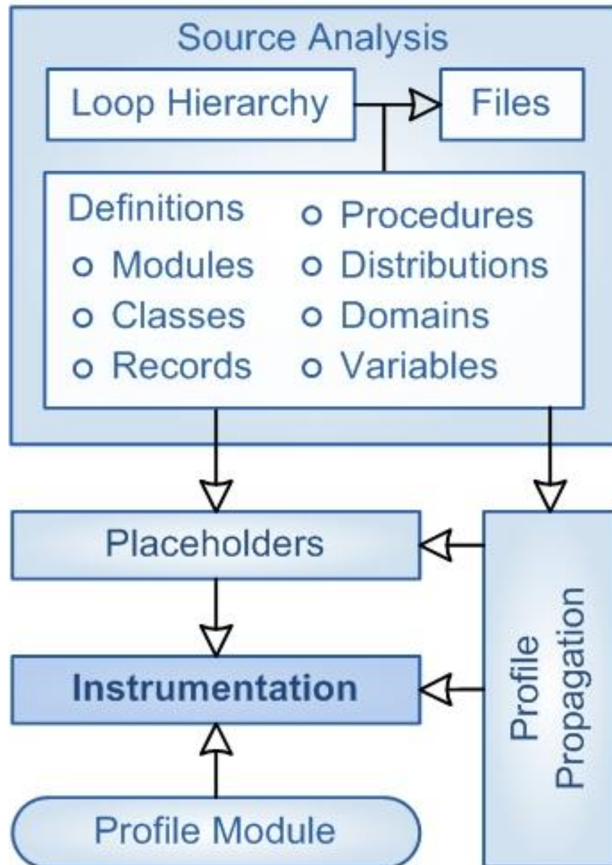
Static Analyzer



- Adds five new passes to compiler

1. Source analysis pass
   - Loads profile module and profile input file
   - Analyzes program ASTs after parser
   - Acquires and stores maps and definitions
2. Instrument variable definition placeholders

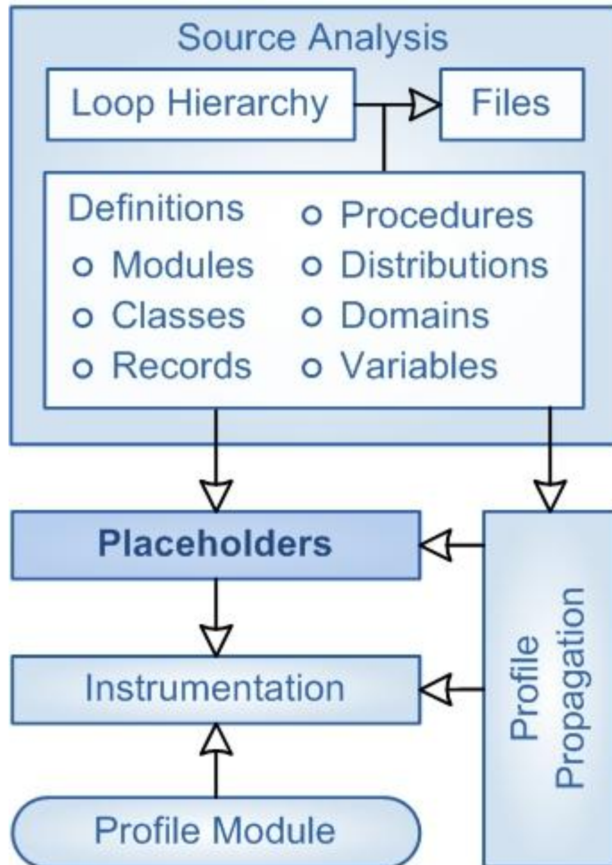
Static Analyzer



- Adds five new passes to compiler

1. Source analysis pass
   - Loads profile module and profile input file
   - Analyzes program ASTs after parser
   - Acquires and stores maps and definitions
2. Instrument variable definition placeholders
3. Instrument compiler generated main
   - Instr. profile load, coordination, and reporting

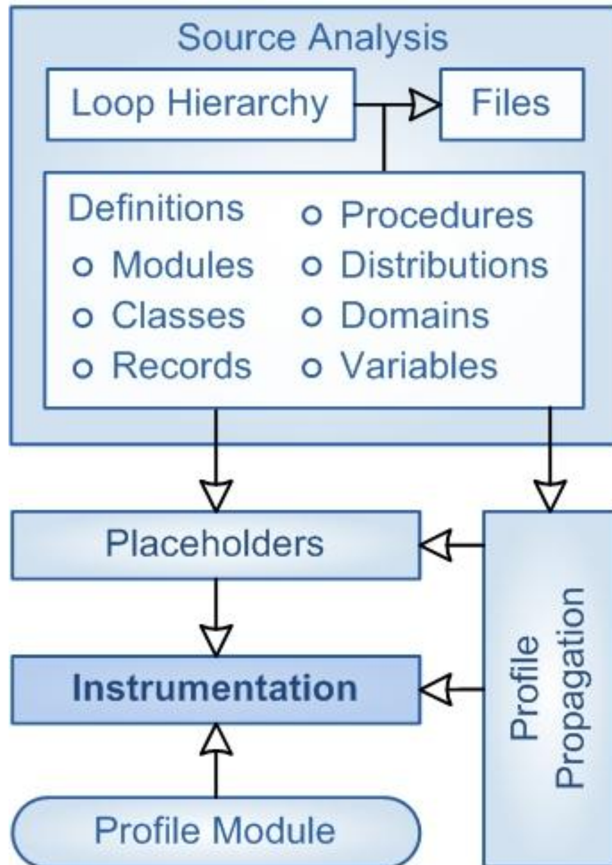
Static Analyzer



- Adds five new passes to compiler

1. Source analysis pass
   - Loads profile module and profile input file
   - Analyzes program ASTs after parser
   - Acquires and stores maps and definitions
2. Instrument variable definition placeholders
3. Instrument compiler generated main
   - Instr. profile load, coordination, and reporting
4. Instrument constructor field placeholders

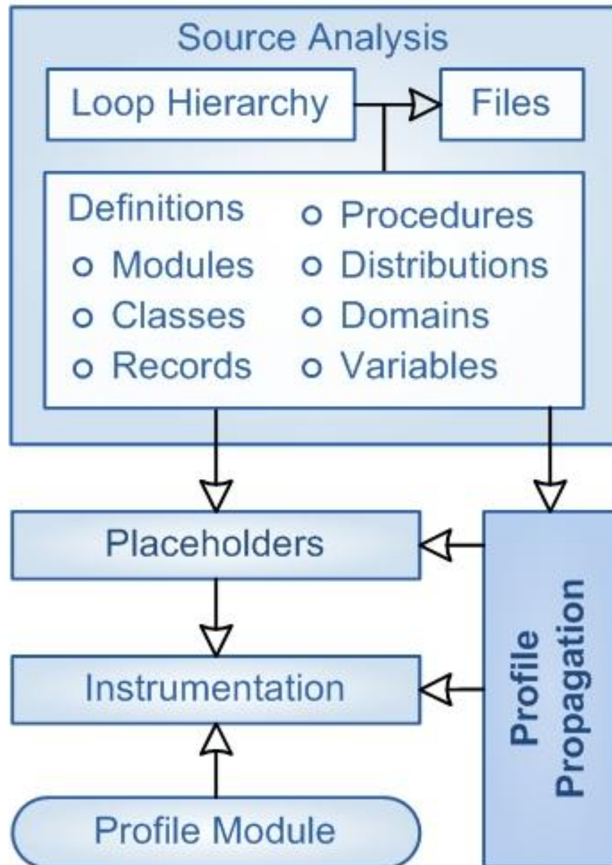
Static Analyzer



- Adds five new passes to compiler

1. Source analysis pass
   - Loads profile module and profile input file
   - Analyzes program ASTs after parser
   - Acquires and stores maps and definitions
2. Instrument variable definition placeholders
3. Instrument compiler generated main
   - Instr. profile load, coordination, and reporting
4. Instrument constructor field placeholders
5. Instrument wide references
   - Placeholder pruning and final instrumentation

Static Analyzer



- Adds five new passes to compiler

1. Source analysis pass
   - Loads profile module and profile input file
   - Analyzes program ASTs after parser
   - Acquires and stores maps and definitions
2. Instrument variable definition placeholders
3. Instrument compiler generated main
   - Instr. profile load, coordination, and reporting
4. Instrument constructor field placeholders
5. Instrument wide references
   - Placeholder pruning and final instrumentation

- Profile propagation
  - Persists profile data across all of the passes

Dynamic Analyzer

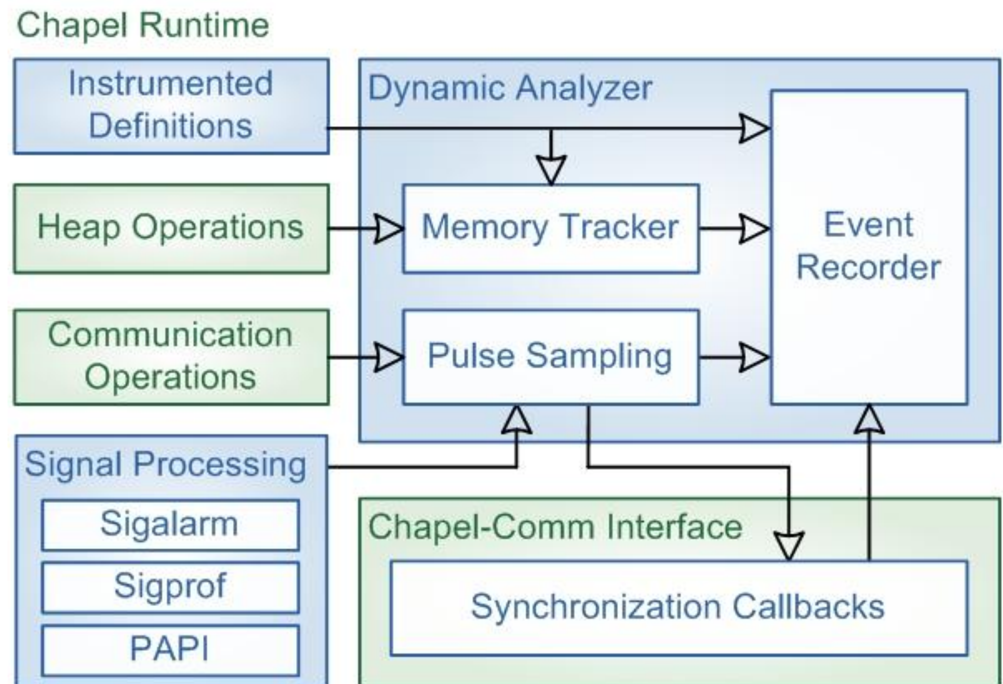
- Dynamic analysis
  - Monitors memory and communication operations
  - Embedded in the Chapel runtime framework
  - Executed through the programs instrumentation
  - Is configurable through environment variables
- Tracking memory
  - Heap operations for variables
    - Chapel allocations, reallocations, and free operations
  - Stack variables
    - Associates address and size with beginning and end of scope

Dynamic Analyzer

- Tracking communications
  - Monitoring
    - Local and remote get and put operations
    - Remote prefetch and cache operations
  - Pulse sampling
    - Samples data at periodic intervals (on / off)
    - Scalable: Reduce runtime overhead and event log size
    - Support: SIGALRM / SIGPROF / PAPI
  - Synchronization
    - Coordinate events in post analysis
    - Support: GASNet, In progress: UGNI

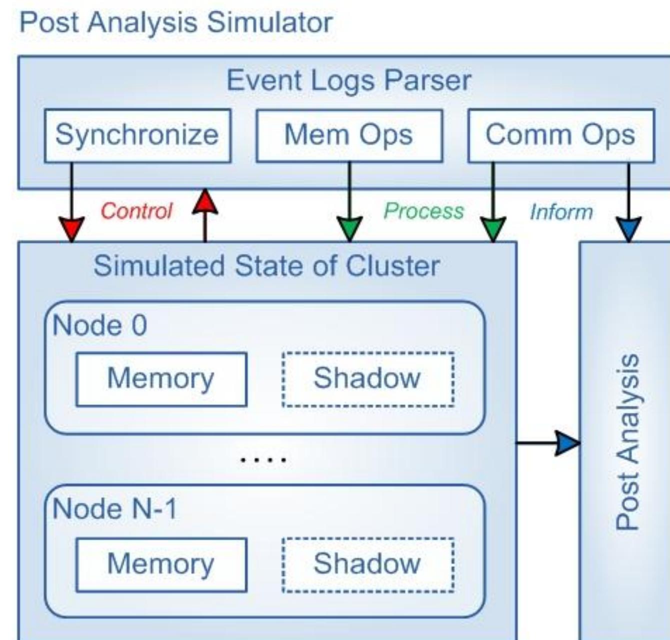
Dynamic Analyzer: Output

- Online reporting
  - High level report of local and remote operations
  - Aggregates over all nodes and entire execution
- Event recorder
  - Each node produces its own event log
  - Used in post analysis
  - What is recorded?
    - Synchronizations
    - Memory operations
    - Communications



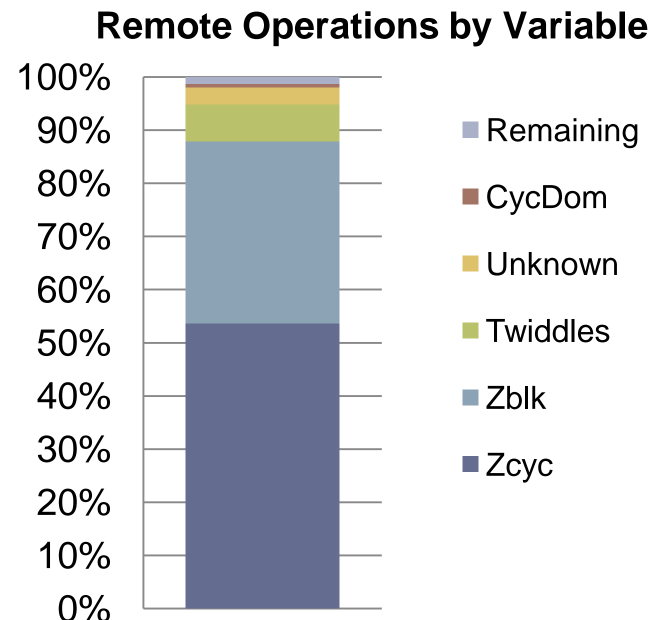
Development: Post Analysis Simulation

- Process event logs into a unified timeline
  - Synchronization: Controls progression of simulation
  - Memory Operations: Update memory environments
  - Communications: Addresses mapped to definitions
    - Unified view of the PGAS



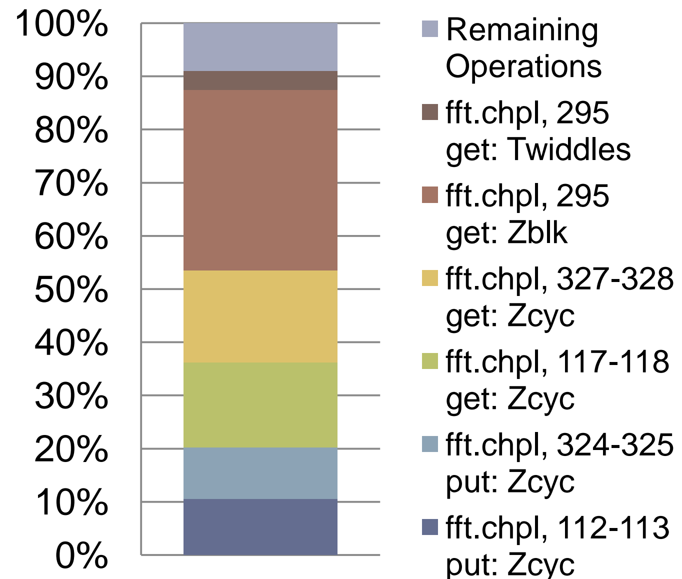
Development: Post Analysis Simulation

- Process event logs into a unified timeline
  - Synchronization: Controls progression of simulation
  - Memory Operations: Update memory environments
  - Communications: Addresses mapped to definitions
    - Unified view of the PGAS
- Analyzer (ranked results)
  - Variable-Cluster Overview



Development: Post Analysis Simulation

- Process event logs into a unified timeline
  - Synchronization: Controls progression of simulation
  - Memory Operations: Update memory environments
  - Communications: Addresses mapped to definitions
    - Unified view of the PGAS
- Analyzer (ranked results)
  - Variable-Cluster Overview
  - Variable-Loop Analysis



Development: Post Analysis Simulation

- Process event logs into a unified timeline
  - Synchronization: Controls progression of simulation
  - Memory Operations: Update memory environments
  - Communications: Addresses mapped to definitions
    - Unified view of the PGAS
- Analyzer (ranked results)
  - Variable-Cluster Overview
  - Variable-Loop Analysis
  - Variable-Node and Index

Accesses to Zcyc[…] by Node

Dst \ Src	n0	n1	n2	n3
n0	1801	464	488	499
n1	515	1332	493	488
n2	506	459	1587	530
n3	506	422	487	1513

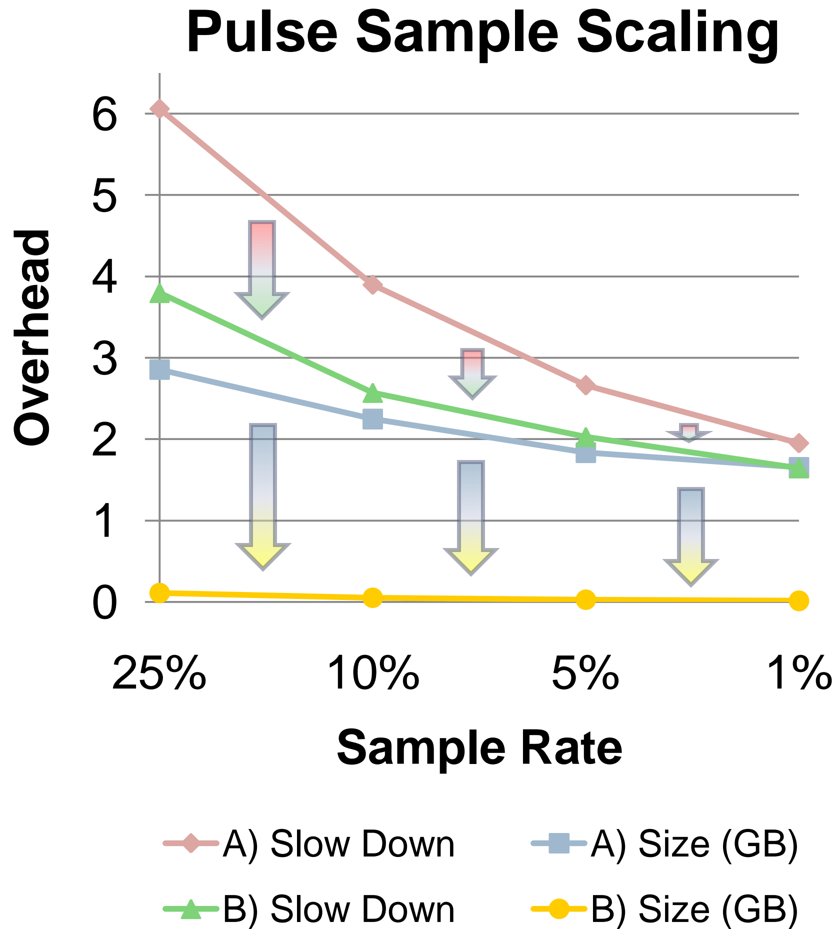
Evaluation

- Deepthought2 HPC cluster
  - Housed at the University of Maryland
  - 444 nodes, 20 cores and 128 GB memory per node
  - FDR infiniband interconnect with 56 Gb/s max throughput
- Chapel configuration
  - GASNet with IBV substrate
  - Using MPI as the spawner
  - Memory segment: ‘everything’
    - All memory is remotely accessible
- Evaluated on SSCA#2 benchmark

SSCA#2: Evaluation

- Description
  - Series of approximate betweeness centrality (BC) calculations
  - Data structure: Weighted, directed multigraph
  - Different input sizes: Sets of starting vertices
- Evaluation
  1) Analyzed overhead and scalability with pulse sampling
  2) Evaluated network caching impact on remote operations
  3) Performed loop analysis and node level aggregation
     - On remote operations to identify PGAS bottlenecks

SSCA#2: Overhead and Scalability

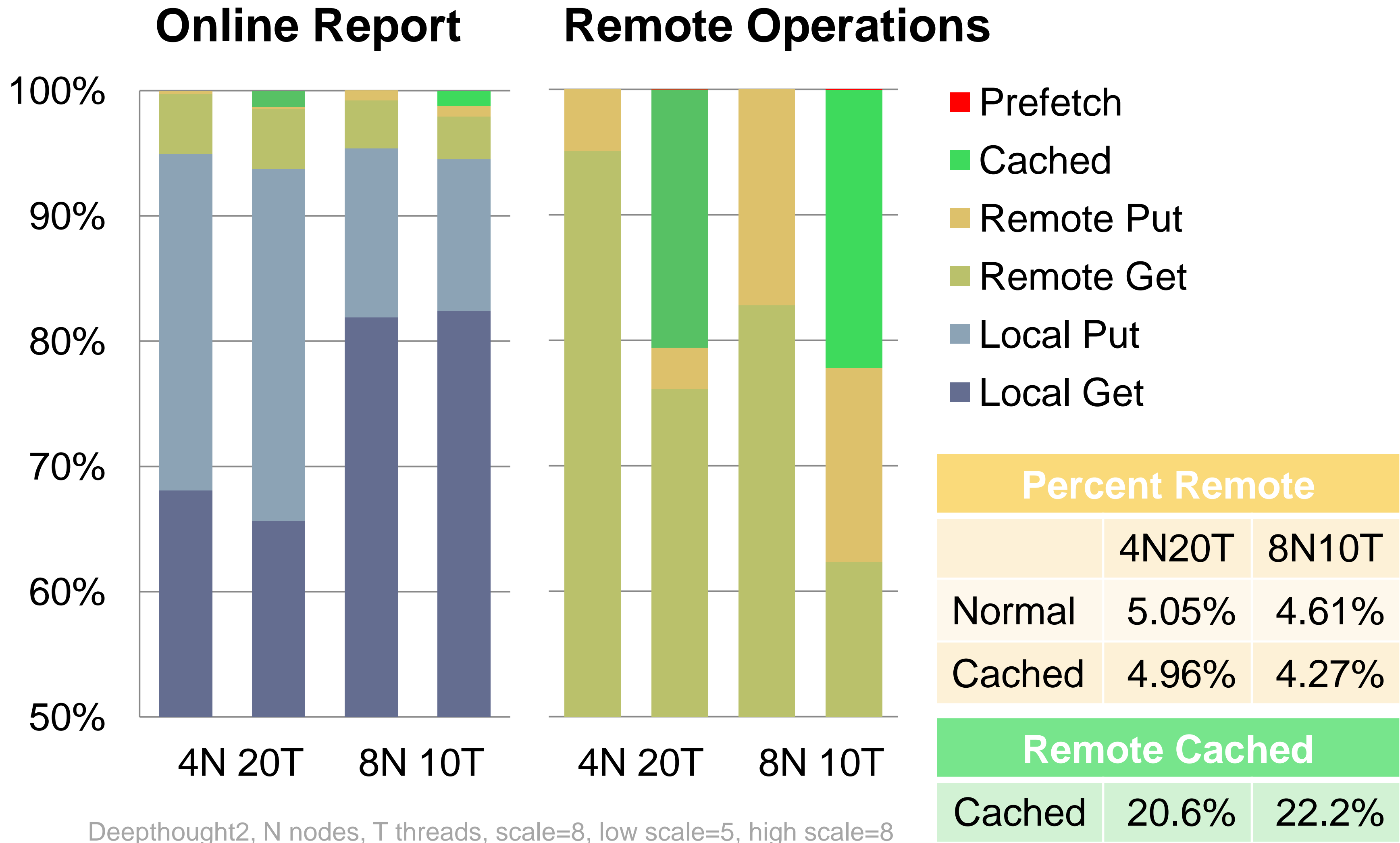


- Tracked all 421 variables
- Online reporting only
  - +16% of wall time
- With event log generation
  - 1% sample rate

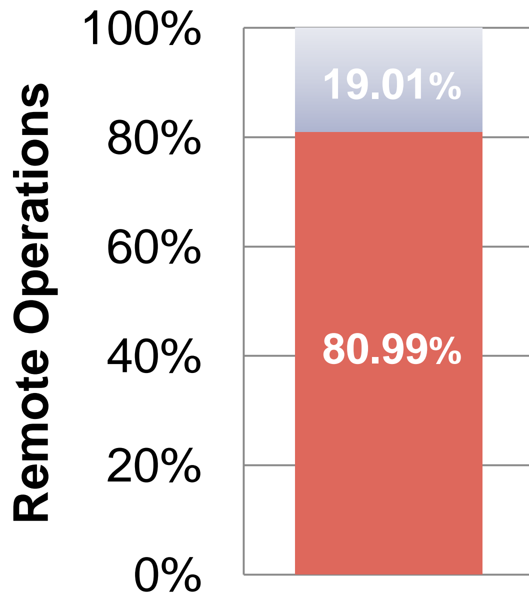
  A. Tracking stack & heap
     - +95% of wall time
     - 1.65 GB for event logs

  B. Tracking heap only
     - +62% of wall time
     - 14.3 MB for event logs

SSCA#2: Network Caching



SSCA#2: Analysis (Remote Operations)



5% sample rate, 98.44% coverage

Deepthought2, 4 nodes, 20 threads, scale=8, low scale=5, high scale=8

SSCA2_kernels.chpl, Approximate Betweeness Centrality

```
279:  forall s in starting_vertices do on
        vertex_domain.dist.idxToLocale(s) {
...
286:    const tid = TPVM.gettid();
287:    const tpv = TPVM.getTPV(tid);
```

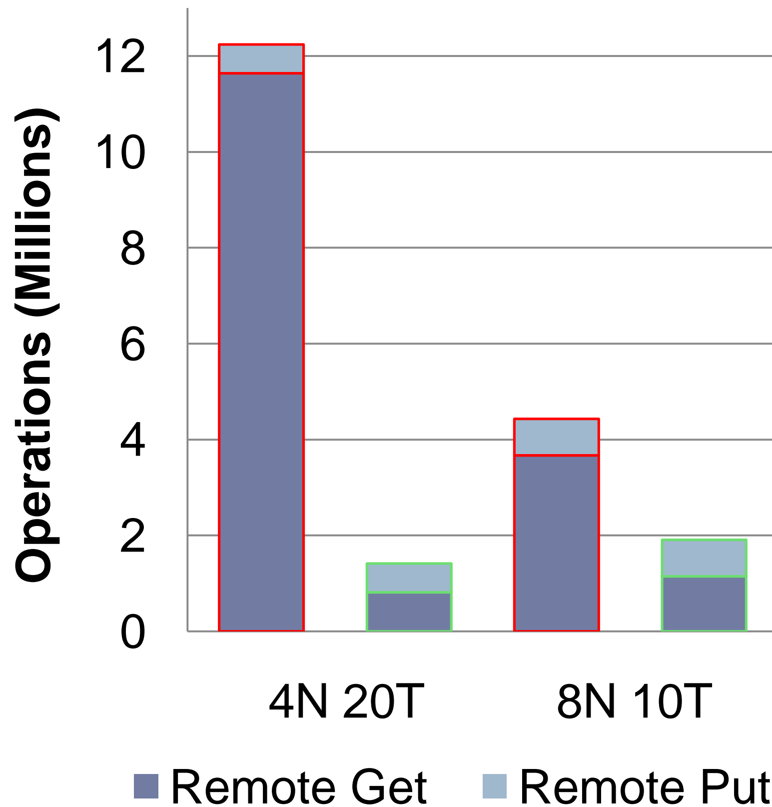
- 80.99% of all remote operations
  - Acquiring 'this.TPV' inside gettid()

Remote Requests for TPVM.TPV

Receiver	Sender n0	Sender n1	Sender n2	Sender n3
n0	0	51256	57384	55021

- Exists only on the primary node

SSCA#2: Optimization



Original: SSCA2_kernel.chpl

```
576: class TPVManager {
579:   proc gettid() {
580:     const tid = this.currTPV.fetchAdd(1) % num…
581:     on this.TPV[tid] do
582:       while this.TPV[tid].used.testAndSet() do
             chpl_task_yield();
583:     return tid;
```

Optimized: SSCA2_kernel.chpl

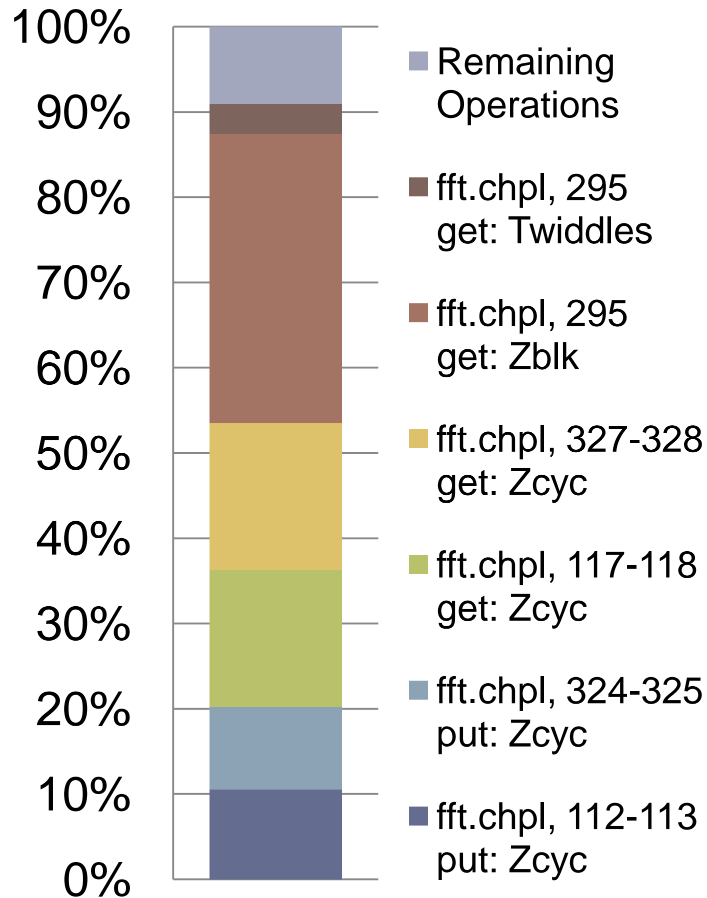
```
576: class TPVManager {
579:   proc gettid() {
580:     const tid = this.currTPV.fetchAdd(1) % num…
581:     on this.TPV[tid] do {
582:       const t = this.TPV[tid]
583:       while t.used.testAndSet() do chpl_task_yiel…
584:     }
585:     return tid;
```

Config	Opt % Remote	Reduction
4N 20T	0.47%	88.45%
8N 10T	1.19%	56.87%

Conclusion

- We developed a data-centric, communication profiler
  - Analyzes memory and communication access patterns
  - Supports multi-node PGAS environments
  - Integrated into the Chapel framework and configurable
  - Handles loop hierarchies and complex data structures
  - Produces online reporting and scalable, fine-grain profiling
- Demonstrated with SSCA#2 benchmark
  - Identified real PGAS bottlenecks
  - Improve developer's understanding of
    - Where PGAS dependencies may exist and
    - How task-data locality behaves in their program
- Improvements due to Purity insights
  - Up to 88% of remote operations where eliminated
  - Up to 1.24x speedup of the program wall time

Loop Analysis Example (HPCC-FFT)



Deepthought2, 4 nodes, 20 threads, n=16

```
293: proc bitReverseShuffle(Vect: [?Dom]) {
294:     const numBits = log2(Vect.numElements),
295:     Perm: [Dom] Vect.eltType = [i in Dom]
             Vect(bitReverse(i, revBits=numBits));
296:     Vect = Perm;
```

33.97% remote: *Zblk* permutations

3.53% remote: *Twiddles* permutations

```
117, 327: forall (b, c) in zip(Zblk, Zcyc) do
118, 328:     b = c;
```

33.25% remote: Mapping *Zcyc* to *Zblk*

```
112, 324: forall (b, c) in zip(Zblk, Zcyc) do
113, 325:     c = b;
```

20.22% remote: Mapping *Zblk* to *Zcyc*

Zcyc, local vs. remote: 82.86% remote