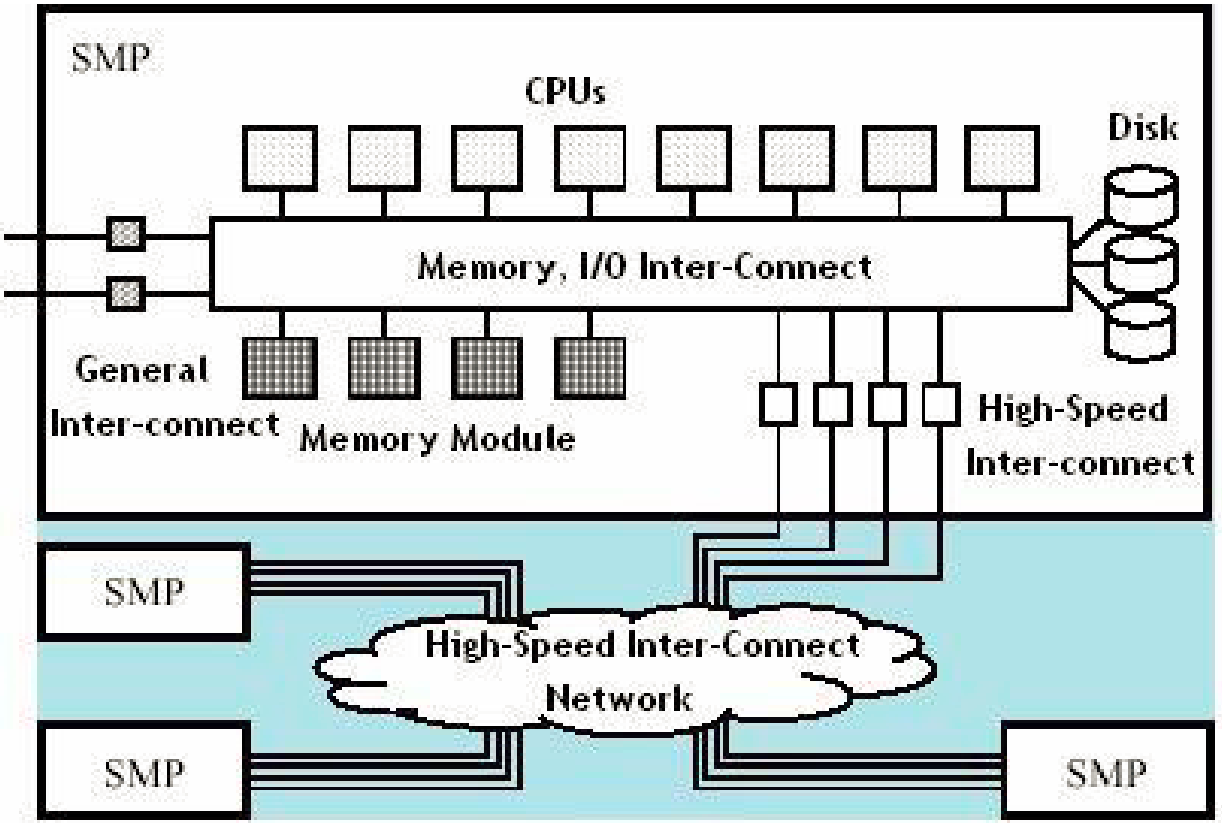



ChplBlamer: A Data-centric and Code-centric Combined Profiler for Multi-locale Chapel Programs

Hui Zhang, Jeffrey K. Hollingsworth
{hzhang86, hollings}@cs.umd.edu
Department of Computer Science, University of Maryland-College Park

Multi-locale Chapel Environment



Motivation

- ***Why PGAS*** (Partitioned Global Address Space)
  - Parallel programming is too hard
  - Unified solution for mixed mode parallelism
- ***Why Chapel***
  - Chapel is an emerging PGAS language with productive parallel programming features
  - Potential for performance improvement (especially in multi-locale) and few Chapel profilers for its users
  - Insights for evolving the language in the future and the same idea can be applied to other parallel programming paradigms through generic approaches

Data-centric Profiling

```
int busy(int *x) {
  // hotspot function
  *x = complex();
  return *x;
}

int main() {
  for (i=0; i<n; i++) {
    A[i] = busy(&B[i]) +
         busy(&C[i-1]) +
         busy(&C[i+1]);
  }
}
```

Code-centric Profiling

main: 100%
busy: 100%
complex: 100%

Data-centric Profiling

A: 100%
B: 33.3%
C: 66.7%

What is “ChplBlamer”?

"MISS HARPER— GET ME SOMEBODY TO BLAME."

Properly Assign Blame

"I didn't say you were to blame… I said I am blaming you."

Blame Definition

1) $\boldsymbol{BlameSet(v) = \bigcup_{w \in W} BackwardSlice(w)}$

2) $\boldsymbol{isBlamed(v, s) = \{if\big(s \in BlameSet(v)\big)\ then\ 1\ else\ 0\}}$

3) $\boldsymbol{BlamePercentage(v, S) = \frac{\sum_{s \in S} isBlamed(v,s)}{|S|}}$

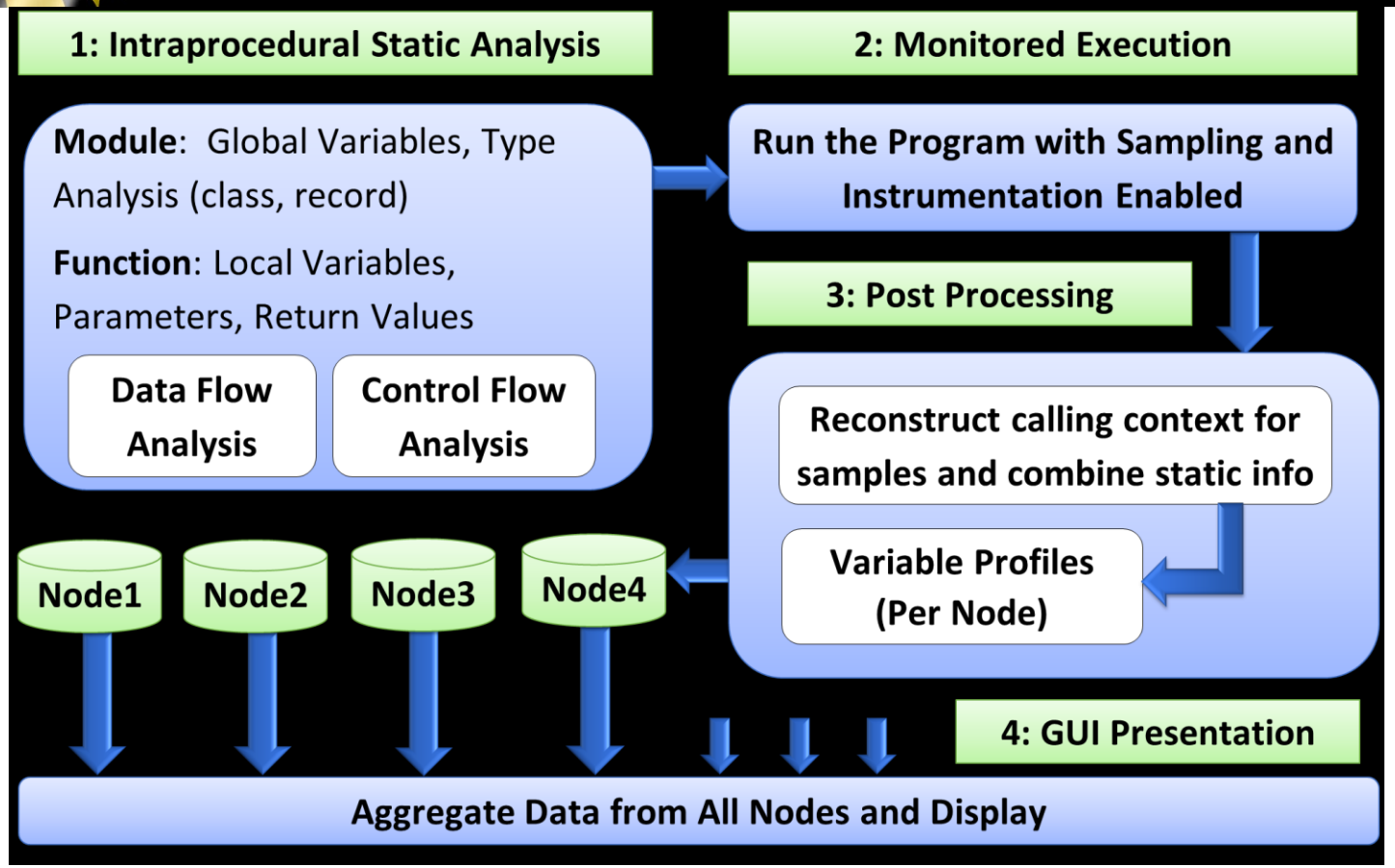
- *v*: a certain variable
- *w*: a write statement to v's memory region
- *W*: a set of w (all write statements to v's memory region)
- *s*: a sample
- *S*: a set of samples

Blame Calculation

```
1    a = 8;                          //Sample 1
2    b = a * a;                      //Sample 2,3
3    for (i = 0; i < N; i++)         //Sample 4
4        b = b + i;
5    c = a + b;                      //Sample 5
```

Variable Name	a		b		c		i	
Result Type	inc	exc	inc	exc	inc	exc	inc	exc
BlameSet	1	1	1,2,3,4	2,4	1,2,3,4,5	5	3	3
Blame Samples	S1	S1	S1,2,3,4	S2,3	S1,2,3,4,5	S5	S4	S4
Blame	20%	20%	80%	40%	100%	20%	20%	20%

ChplBlamer Framework



[1] Zhang, Hui, and Jeffrey K. Hollingsworth. "Data Centric Performance Measurement Techniques for Chapel Programs." Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International. IEEE, 2017.

Multi-locale Challenges

- **1st Challenge:**

Aggregate blame of many temporary variables that point/refer to the distributed variables through remote data accesses.

- **Solution**:
  - Link variable PvID (privatized id) with different objects accessed through specifc Chapel runtime functions: *chpl_getPrivatizedCopy,* and *chpl_getPrivatizedClass.*

Multi-locale Challenges

- **2nd Challenge:**
  - Recover the **hidden** data-flow information from Chapel internal module calls, e.g., chpl_gen_comm_get
  - Recover the **interrupted** data-flow information from Chapel runtime calls, e.g., chpl_*taskListAddBegin*
- **Solution**:
  - Conduct simplified blame analysis for Chapel module functions to get data-dependencies between parameters
  - Resolve actual wrapper task function statically through function pointers that were passed to certain Chapel runtime functions

Multi-locale Challenges

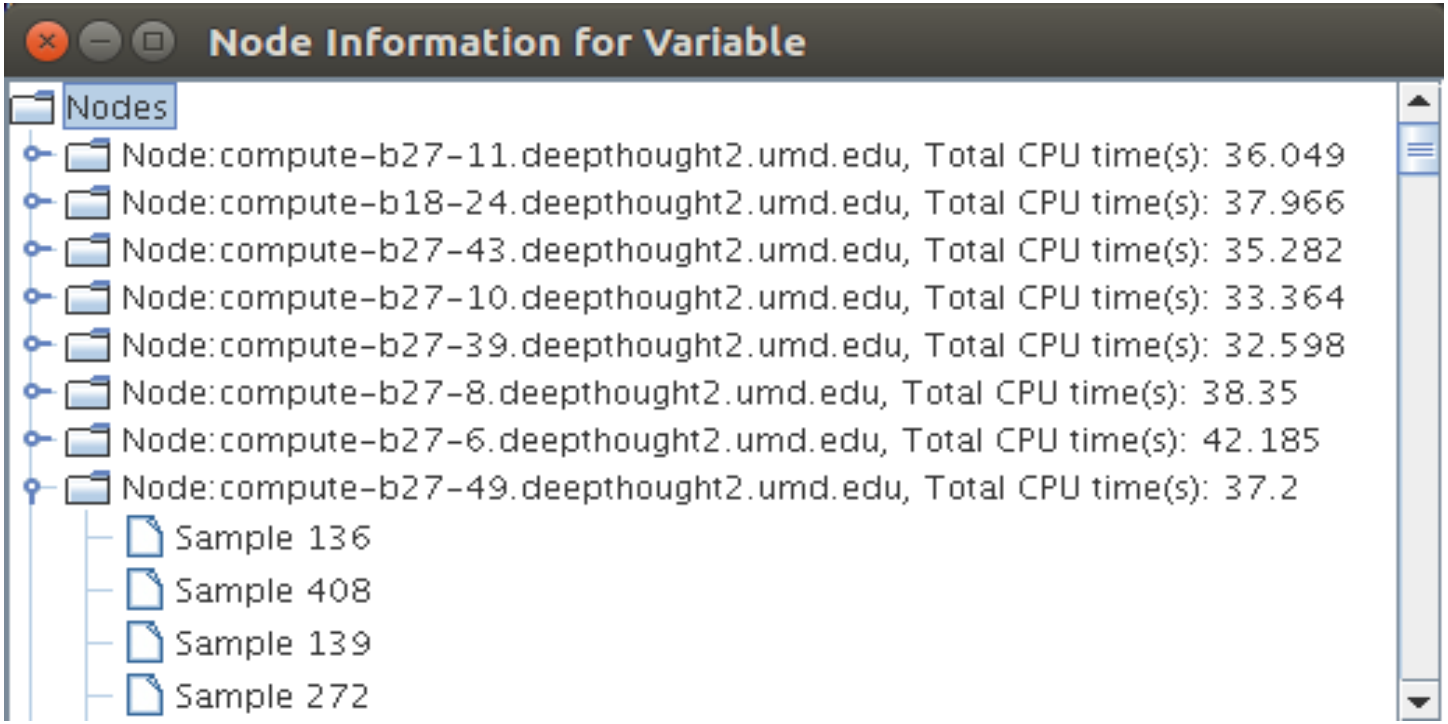
- **3rd Challenge:**

Reconstruct the full calling context for each sample and handle asynchronous&remote tasking

- **Solution**:

- Instrument Chapel tasking and communication layer
- Log "***task function ID***", "***task sender's locale ID", and "task receiver's locale ID"*** for each remote task
- Iteratively glue stacktraces to the current calling context until having the user "***main***" frame

New Tool Feature
Load Imbalance Check



Node information for *Ab* of HPL on 32 locales

Experiment – ISx

Data-centric	2-loc	8-loc
myBucketedKeys	41.1%	22.9%
myKeys	36.9%	20.9%
sendOffsets	27.3%	15.4%
bucketOffsets	26.9%	15.2%
barrier	10.3%	20.8%

Code-centric	2-loc	8-loc
bucketSort	80.9%	64.2%
bucketizeLocalKeys	40.2%	22.3%
countLocalKeys	11.4%	6.4%
pthread_spin_lock	16.7%	29.3%
chpl_comm_barrier	0	3.46%

Name	original	localization
myBucketedKeys	41.11%	17.78%
sendOffsets	27.28%	6.02%
bucketOffsets	26.85%	5.46%
bucketizeLocalKeys	40.24%	24.54%

OPTIMIZATION:

1. Optimize "Barrier" module
2. Apply "local" clause

Experiment - LULESH

Variable	Type	Blame	Context
Elems	Struct	74.3%	chpl_gen_main
elemToNode	Struct	60.4%	chpl_gen_main
xd/yd/zd	Struct	48.0%	chpl_gen_main
x/y/z	Struct	37.0%	chpl_gen_main
fx/fy/fz	Struct	35.6%	chpl_gen_main
dvdx/dvdy/dvdz	Struct	33.4%	CalcHourglassControlForElems
x8n/y8n/z8n	Struct	33.3%	CalcHourglassControlForElems
elemMass	Struct	29.5%	chpl_gen_main
hgfx/hgfy/hgfz	Array	26.7%	CalcFBHourglassForceForElems
shx/shy/shz	Double	26.7%	CalcElemFBHourglassForce
hx/hy/hz	Array	26.6%	CalcElemFBHourglassForce
dxx/dyy/dzz	Struct	12.2%	CalcLagrangeElements

LULESH Optimization: Globalization

Variable	Blame	Context
Elems	74.3%	chpl_gen_main
elemToNode	60.4%	chpl_gen_main
xd/yd/zd	48.0%	chpl_gen_main
x/y/z	37.0%	chpl_gen_main
fx/fy/fz	35.6%	chpl_gen_main
dvdx/dvdy/dvdz	33.4%	CalcHourglassControlForElems
x8n/y8n/z8n	33.3%	CalcHourglassControlForElems
elemMass	29.5%	chpl_gen_main
hgfx/hgfy/hgfz	26.7%	CalcFBHourglassForceForElems
shx/shy/shz	26.7%	CalcElemFBHourglassForce
hx/hy/hz	26.6%	CalcElemFBHourglassForce
dxx/dyy/dzz	12.2%	CalcLagrangeElements

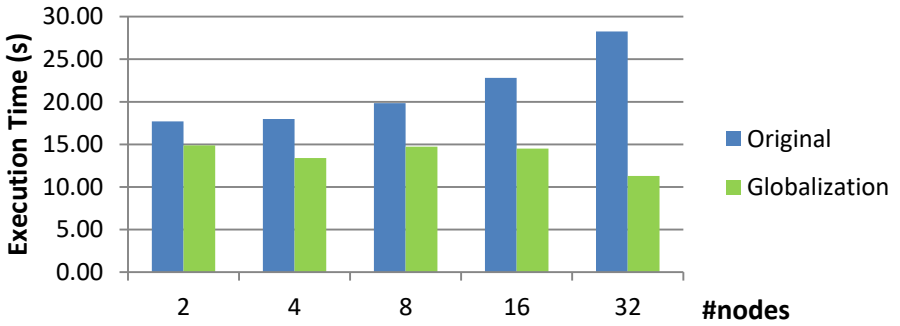
Problem:

```
proc CalcHourglassControlForElems (determ) {
    var dvdx, dvdy, dydz, x8n, y8n, z8n: [Elems] 8*real;
...
```

Solution:

Hoisting distributed local variables to the global space so that they won't be dynamically allocated frequently.

Result:



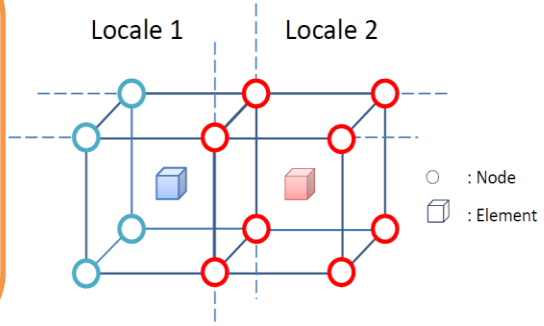
LULESH Optimization: Replication

Variable	Blame	Context
Elems	74.3%	chpl_gen_main
elemToNode	60.4%	chpl_gen_main
xd/yd/zd	48.0%	chpl_gen_main
x/y/z	37.0%	chpl_gen_main
fx/fy/fz	35.6%	chpl_gen_main
dvdx/dvdy/dvdz	33.4%	CalcHourglassControlForElems
x8n/y8n/z8n	33.3%	CalcHourglassControlForElems
elemMass	29.5%	chpl_gen_main
hgfx/hgfy/hgfz	26.7%	CalcFBHourglassForceForElems
shx/shy/shz	26.7%	CalcElemFBHourglassForce
hx/hy/hz	26.6%	CalcElemFBHourglassForce
dxx/dyy/dzz	12.2%	CalcLagrangeElements

Problem:

Frequent calls to "*localizeNeighborNodes* " on these variables which incurs sequential remote data accesses.

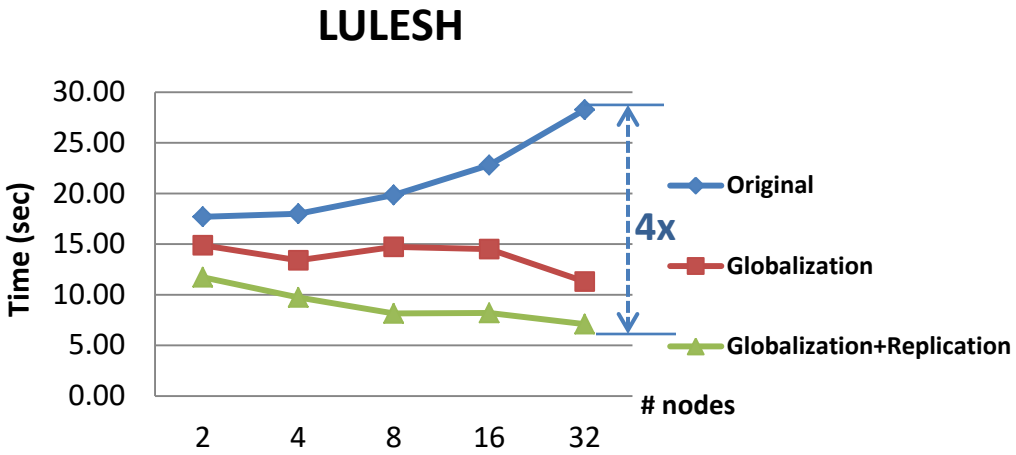
```
for i in 1..nodesPerElem
{
  const noi =
  elemToNode[eli][i];
  x_local[i] = x[noi];
  y_local[i] = y[noi];
  z_local[i] = z[noi];
}
```



Solution:

Allocate global maps to prestore neighboring nodes for each element using the same domain: *var x_map: [Elems] nodesPerElem*real*

Conclusion



move from having slowdown as more locales were added to having speedups!

- Data-centric Profiling and Blame Analysis
- Multi-locale Support and New Features
- Benchmark Profiling and Optimization