Implementation of a Multi-locale Chapel Profiler

Hui Zhang, Jeffrey K. Hollingsworth

{hzhang86, hollings}@cs.umd.edu

Department of Computer Science, University of Maryland-College Park

Motivation

- Chapel is an emerging PGAS language with productive parallel programming features
- Potential for performance improvement (especially in multi-locale) and few Chapel-specific profilers for its end users
- Insights for the language evolvement in the future and same idea can be applied to other parallel programming paradigms

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%

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** and **interrupted** data-flow information from Chapel runtime and internal module function calls (chpl_gen_comm_get, chpl_*taskListAddBegin, etc.*)

- **Solution**: Conduct simplified blame analysis for Chapel standard modules; resolve actual wrapper task function statically through function pointers

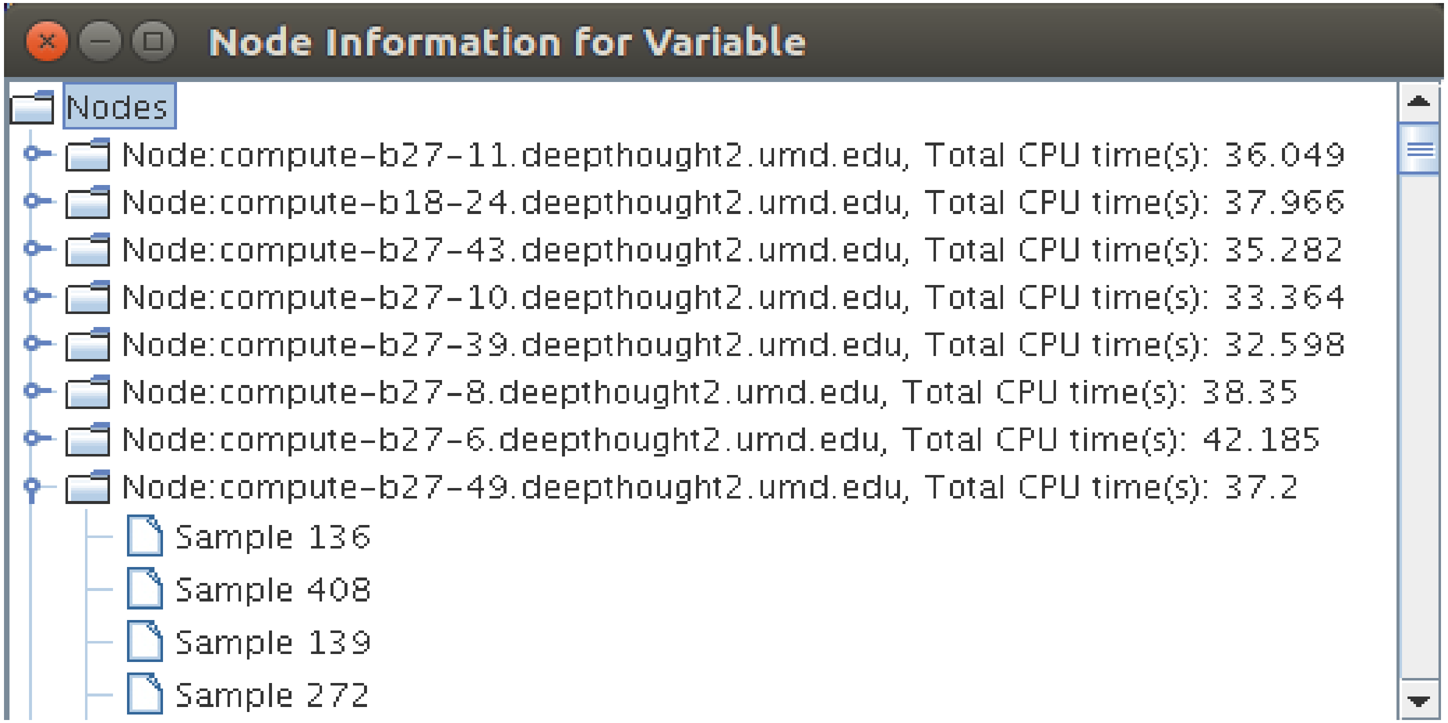
Multi-locale Challenges

- **3rd Challenge:**

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

- **Solution**: Instrument Chapel tasking and communication layer; log “fID, sID and rID” for each remote task; iteratively glue stacktraces before the current calling context until “main”

New Tool Functionality
Load Imbalance Check



```
Nodes
  Node:compute-b27-11.deepthought2.umd.edu, Total CPU time(s): 36.049
  Node:compute-b18-24.deepthought2.umd.edu, Total CPU time(s): 37.966
  Node:compute-b27-43.deepthought2.umd.edu, Total CPU time(s): 35.282
  Node:compute-b27-10.deepthought2.umd.edu, Total CPU time(s): 33.364
  Node:compute-b27-39.deepthought2.umd.edu, Total CPU time(s): 32.598
  Node:compute-b27-8.deepthought2.umd.edu, Total CPU time(s): 38.35
  Node:compute-b27-6.deepthought2.umd.edu, Total CPU time(s): 42.185
  Node:compute-b27-49.deepthought2.umd.edu, Total CPU time(s): 37.2
      Sample 136
      Sample 408
      Sample 139
      Sample 272
```

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

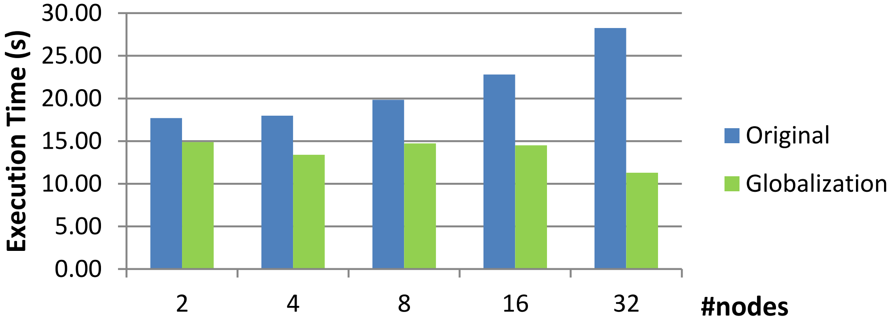
Problem:

```
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:



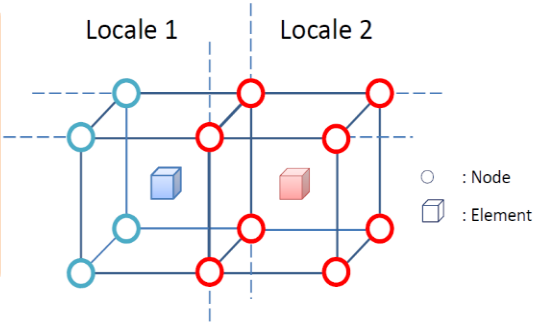
LULESH Optimization: Replication

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.

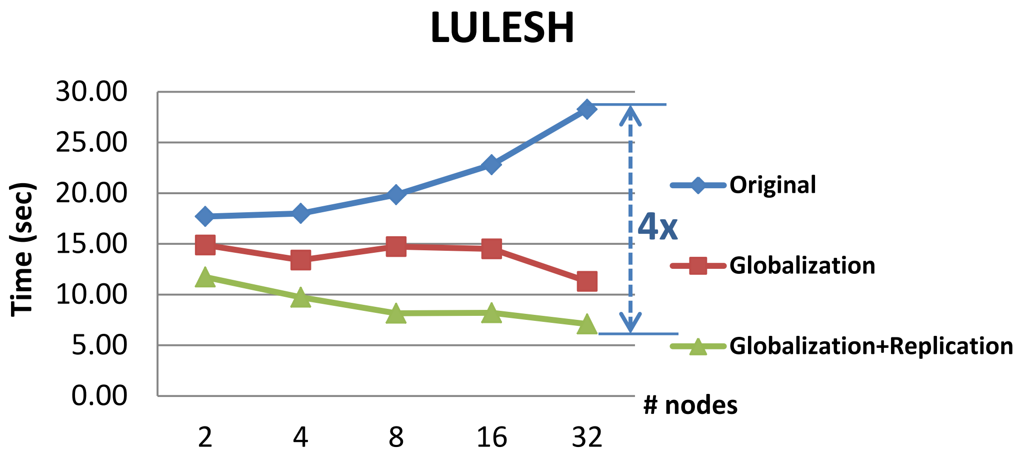
```
for i in 1..nodesPerElem
{
   const noi  =
   elemToNode[eli][i];
   x_local[i] = x[noi];
   y_local[i] = y[noi];
   z_local[i] = z[noi];
}
```



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
- Full paper will be published at ICS’18

(“ChplBlamer: A Data-centric and Code-centric Combined Profiler for Multi-locale Chapel Programs”)