Data-Centric Performance Measurement Technique for Chapel Programs

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

Introduction

- **Why PGAS** (Partitioned Global Address Space)
  - Parallel programming is too hard
  - Unified solution for mixed mode parallelism (multi-core + multi-node)
- **Why Chapel**
  - Emerging PGAS language with productive features
  - Potential for performance improvement and few useful profilers for its end users
  - Insights for the language evolvement in the future

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% latency
busy: 100% latency
complex: 100% latency

Data-centric Profiling

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

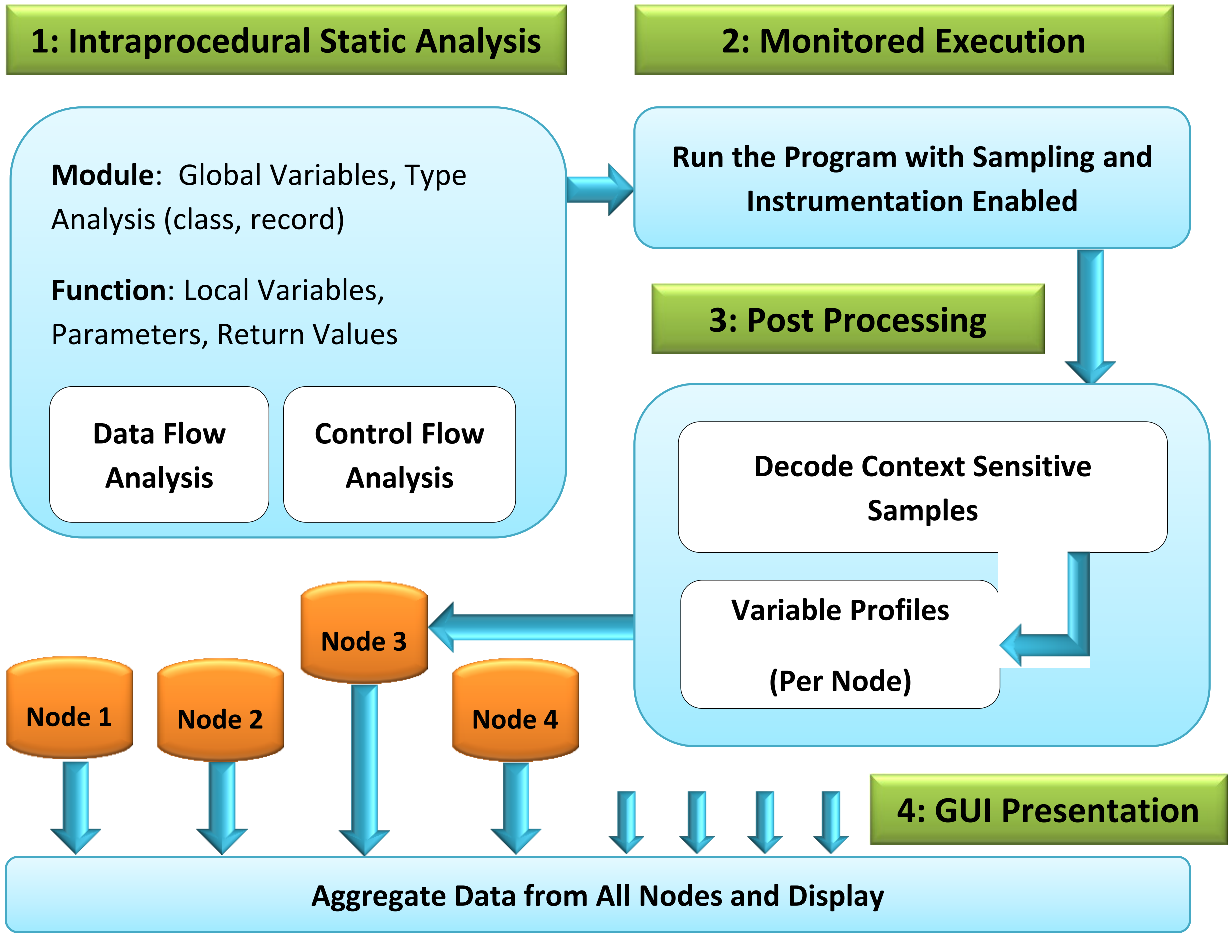
Our Contribution

1. **Data-centric profiling of PGAS programs**

2. **First Chapel-specific profiler**

3. **Profiled three benchmarks and improved the performance up to 2.3x**

Tool Framework



Blame Definition

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

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

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

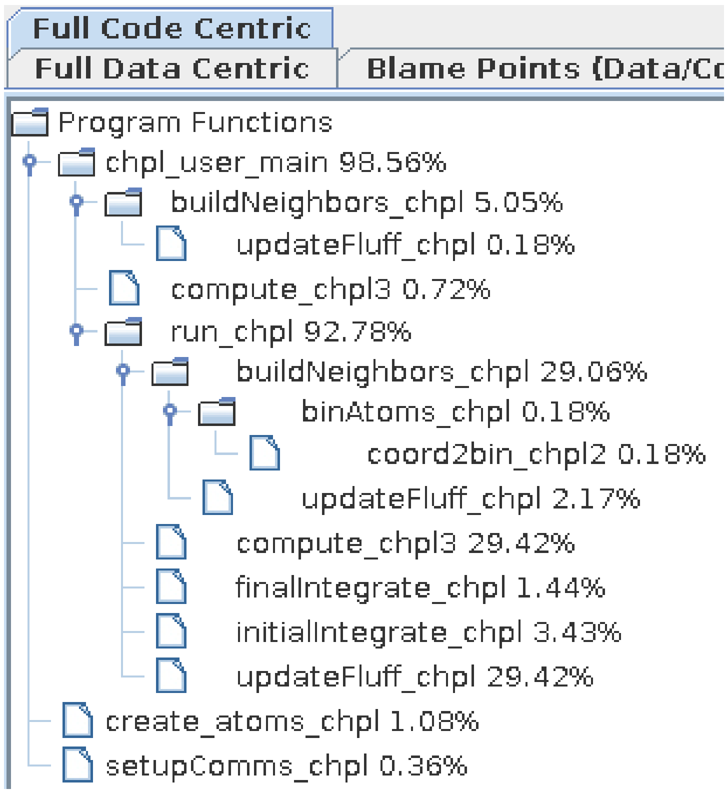
Blame Calculation Example

```
1    a=2;
2    b=3;        //Sample 1
3    if a<b      //Sample 2
4      a=b+1;    //Sample 3
5    c=a+b;      //Sample 4
```

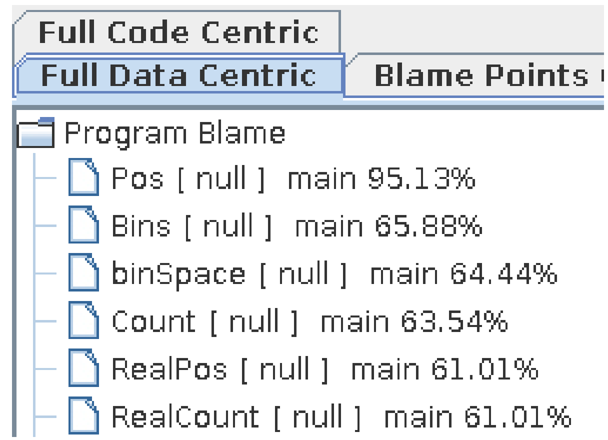
Variable Name	a	b	c
BlameSet	1, **3, 4**	**2**	1, **2, 3, 4, 5**
Blame Samples	S2, S3	S1	S1, S2, S3, S4
Blame	50%	25%	100%

GUI screenshots of MiniMD

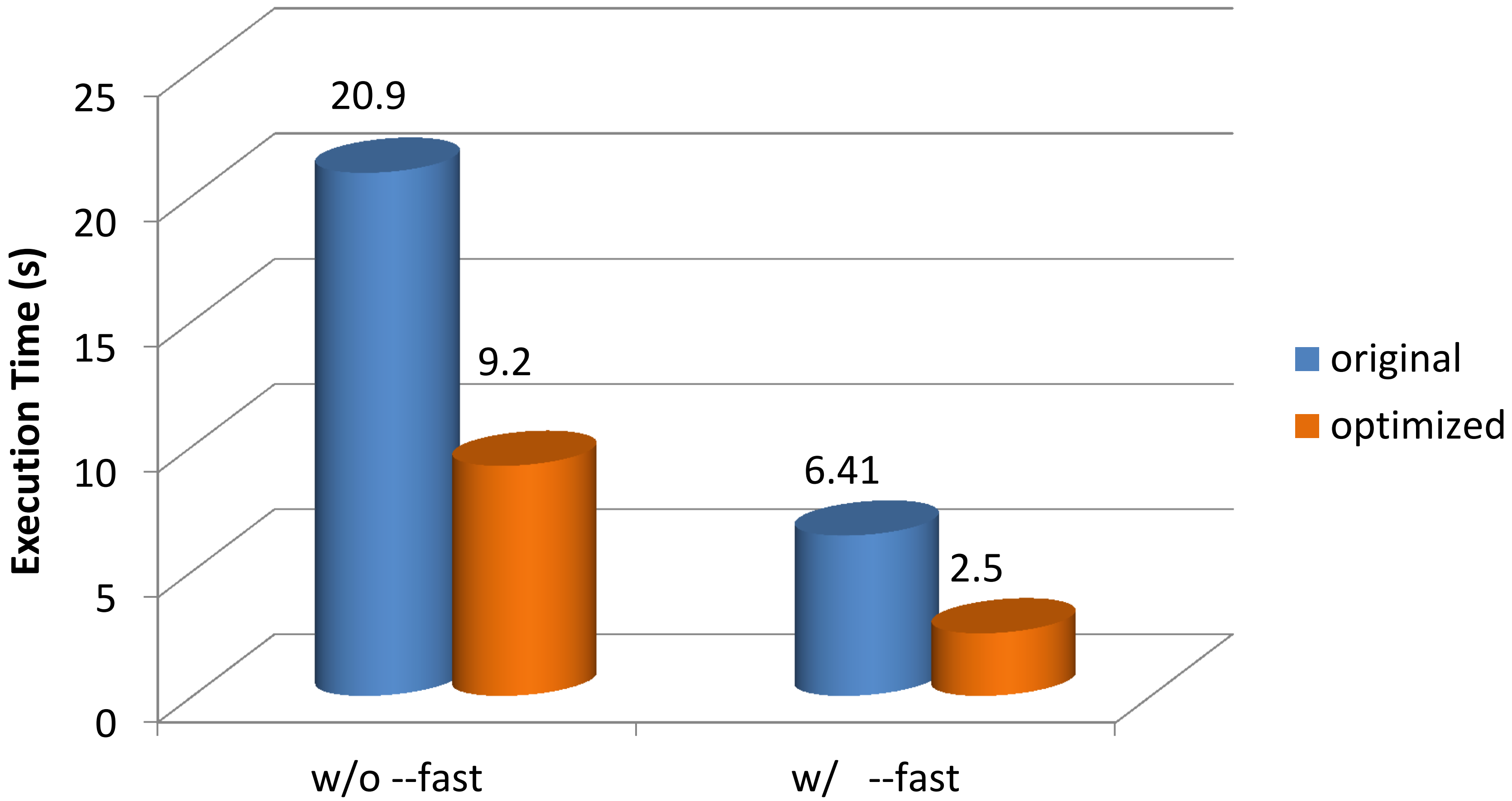
Code-centric



Data-centric



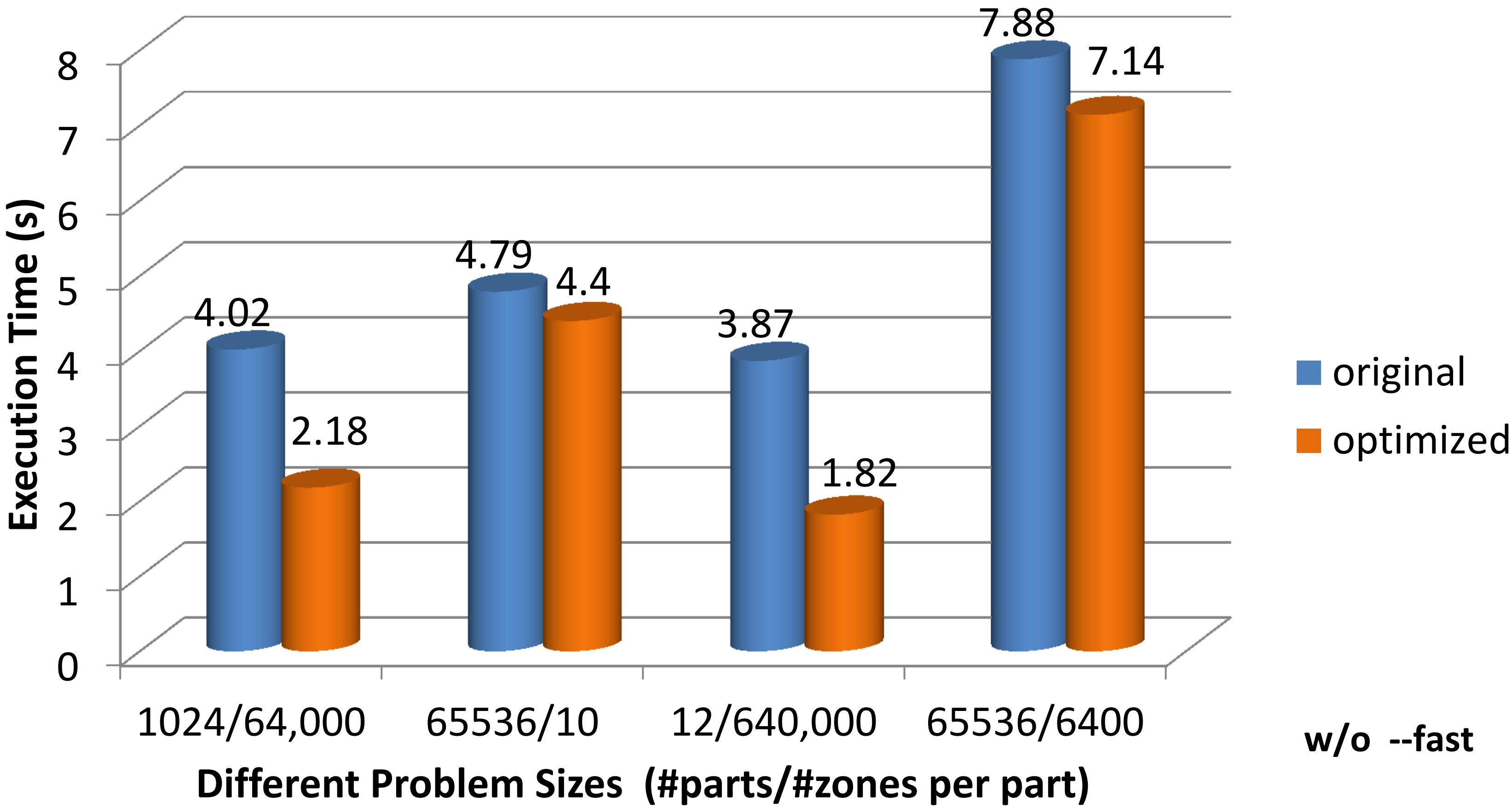
Optimization Result - MiniMD



Experiment - CLOMP

Name	Type	Blame	Context
partArray	[partDomain] Part	99.5%	main
->partArray[i]	Part	99.5%	main
->partArray[i].zoneArray[j]	Zone	99.0%	main
->partArray[i].zoneArray[j].value	real	99.0%	main
->partArray[i].residue	real	12.3%	main
remaining_deposit	real	11.8%	update_part

Optimization Result – CLOMP



w/o --fast

Experiment – LULESH

```
Using local file ./lulesh.
Using local file prof.log.
Total: 17947 samples
```

1	2	3	4	5	6
14180	79.0%	79.0%	14180	79.0%	__sched_yield
834	4.6%	83.7%	943	5.3%	coforall_fn_chpl22
694	3.9%	87.5%	694	3.9%	__pthread_setcancelstate
216	1.2%	88.7%	216	1.2%	atomic_fetch_add_explicit__real64
163	0.9%	89.6%	164	0.9%	coforall_fn_chpl38
160	0.9%	90.5%	272	1.5%	CalcElemNodeNormals_chpl
143	0.8%	91.3%	291	1.6%	coforall_fn_chpl31
123	0.7%	92.0%	586	3.3%	coforall_fn_chpl19
104	0.6%	92.6%	104	0.6%	chpl_thread_yield
95	0.5%	93.1%	95	0.5%	_init

1. Number of profiling samples in this function
2. Percentage of profiling samples in this function
3. Cumulative percentage of samples
4. Number of samples in this function and its callees
5. Percentage of samples in this function and its callees
6. Function name

Experiment – LULESH

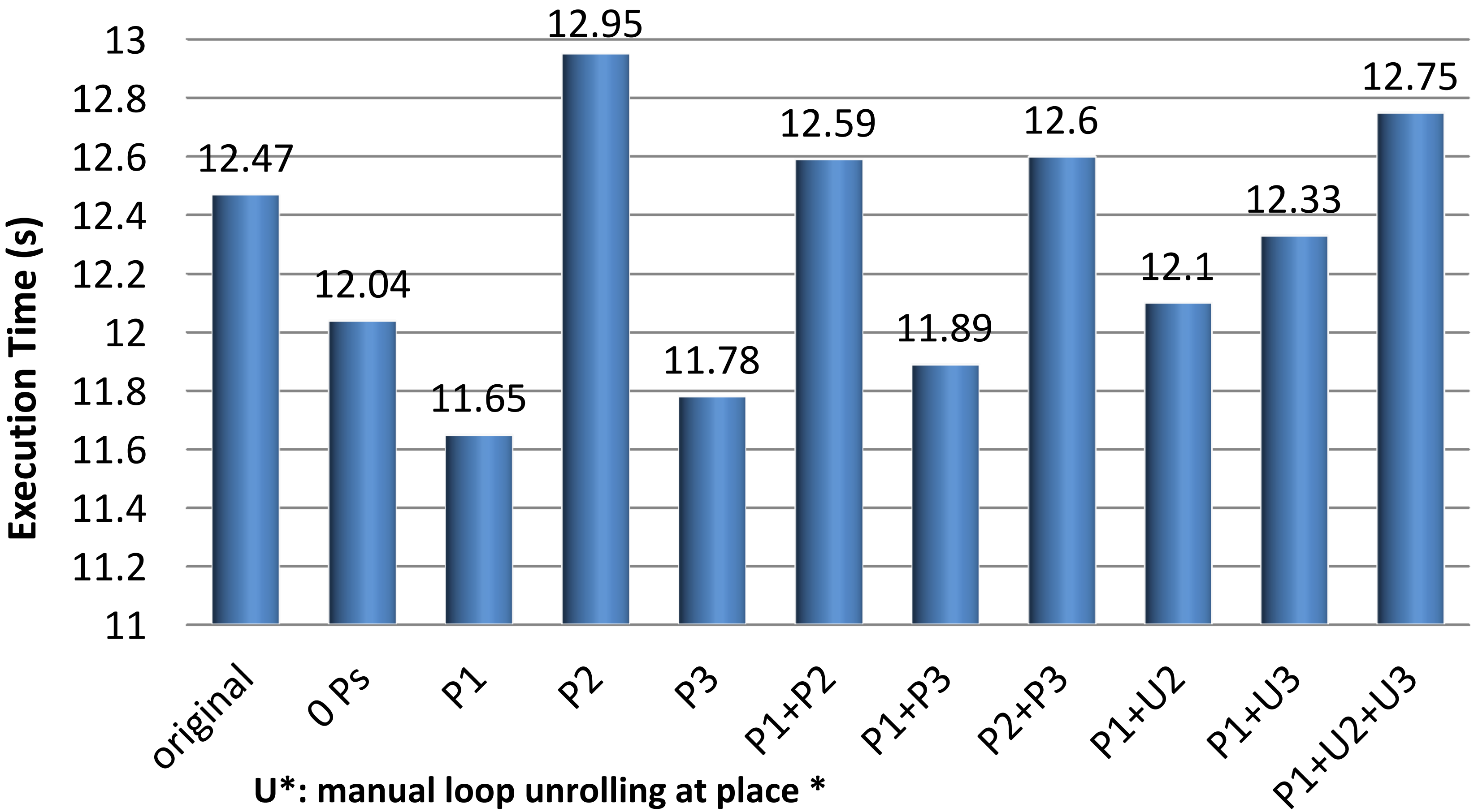
Name	Type	Blame	Context
hgfz	8*real	30.8%	CalcFBHourglassForceForElems
hgfx	8*real	29.5%	CalcFBHourglassForceForElems
hgfy	8*real	29.2%	CalcFBHourglassForceForElems
shz	real	27.9%	CalcElemFBHourglassForce
hz	4*real	27.6%	CalcElemFBHourglassForce
shx	real	26.9%	CalcElemFBHourglassForce
shy	real	26.6%	CalcElemFBHourglassForce
hx	4*real	26.6%	CalcElemFBHourglassForce
hy	4*real	26.6%	CalcElemFBHourglassForce
hourgam	8*(4*real)	25.0%	CalcFBHourglassForceForElems
determ	[Elems] real	15.7%	CalcVolumeForceForElems
b_x	8*real	9.7%	IntegrateStressForElems
b_z	8*real	9.7%	IntegrateStressForElems
b_y	8*real	8.7%	IntegrateStressForElems
dvdx(y/z)	[Elems] 8*real	8.3%	CalcHourglassControlForElems
hourmodx	real	5.8%	CalcFBHourglassForceForElems
hourmody	real	5.1%	CalcFBHourglassForceForElems
hourmodz	real	4.8%	CaclFBHourglassForceForElems

Optimization Example - Loop

```
for param i in 1..4 {        //P1
  var hourmodx, hourmody, hourmodz: real;
  // reduction
  for param j in 1..8 {      //P2
    hourmodx += x8n[eli][j] * gammaCoef[i][j];
    hourmody += y8n[eli][j] * gammaCoef[i][j];
    hourmodz += z8n[eli][j] * gammaCoef[i][j];
  }
  for param j in 1..8 {      //P3
    hourgam[j][i] = gammaCoef[i][j] - volinv *
        (dvdx[eli][j] * hourmodx +
         dvdy[eli][j] * hourmody +
         dvdz[eli][j] * hourmodz);
  }
}
```

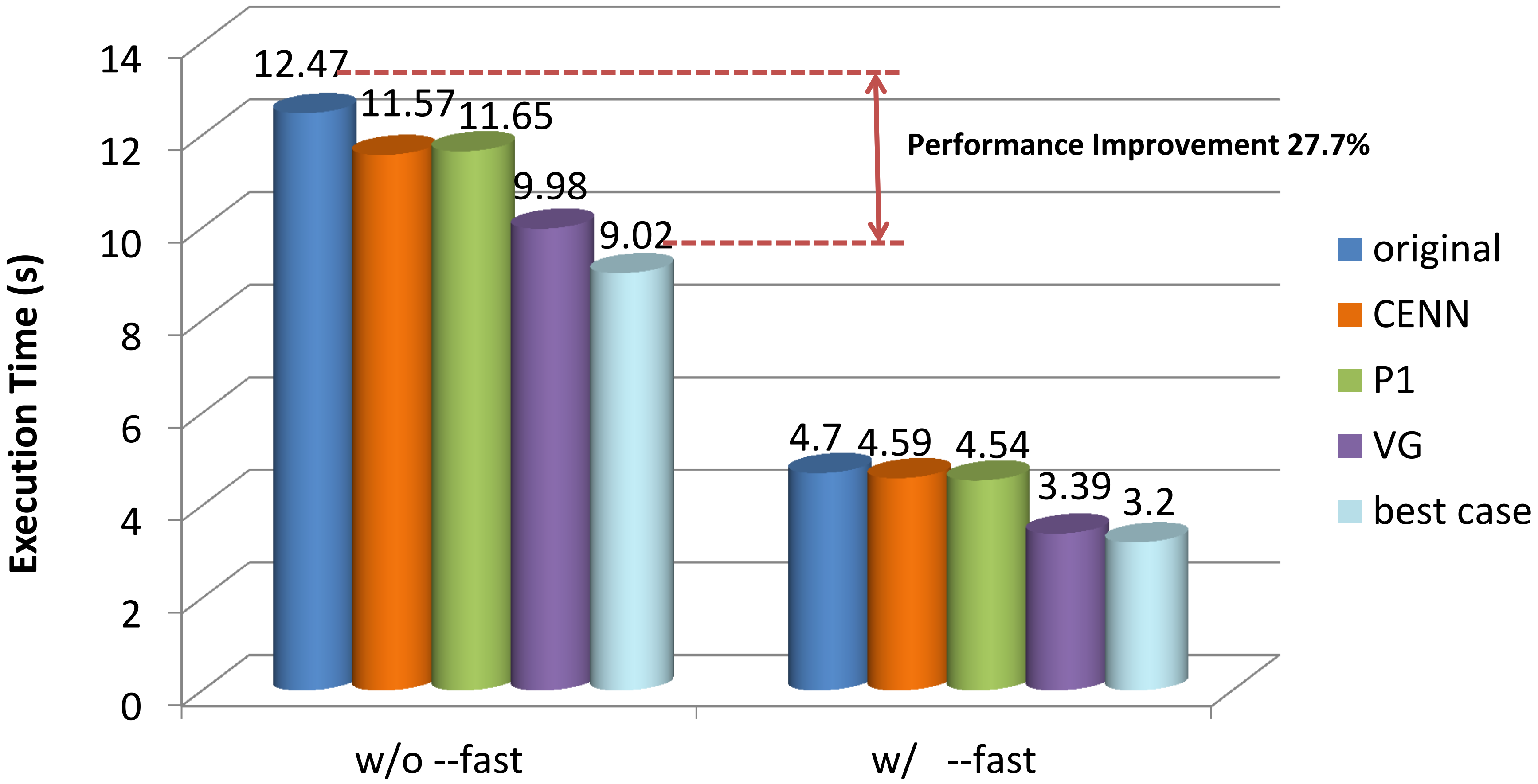
Code Snapshot of LULESH Hot Spot

Results for different loop optimizations



U*: manual loop unrolling at place *

Optimization Result – LULESH



Updates & Future Work

- **Updates**:
  - Built a prototype for multi-node Chapel
  - Optimized runtime instrumentation
  - Improved Graphic-User-Interface
- **Future work:**
  - Large-size problems on distributed systems
  - Further application of “Blame” in other fields

Conclusion

- "Blame" application on PGAS programs
- First Chapel-specific profiler
- Benchmark optimization