

# Performance Analysis and Projections

Ronak Buch

20 April 2016



Charm++

“We must not allow the clock and the calendar to blind us to the fact that each moment of life is a miracle and mystery.”

*H.G. Wells*

Each moment of life may  
be a mystery, but each  
moment of our programs  
shouldn't be.

# Measuring Performance

In the serial world:

- `clock_gettime()`, `gettimeofday()`
- `gprof`
- `valgrind`

# Measuring Performance

In the serial world:

- `clock_gettime()`, `gettimeofday()`
- `gprof`
- `valgrind`

## Parallel

These techniques don't really work for distributed parallel programs.

# Problems in Parallel Analysis

- Different processors may do radically different work
- Performance problems may be on the network
- Load may not be balanced between processors

# Parallel Analysis Tools

In general, tools must be used to do parallel performance analysis:

- mpiP
- Vampir
- HPCToolkit
- Projections

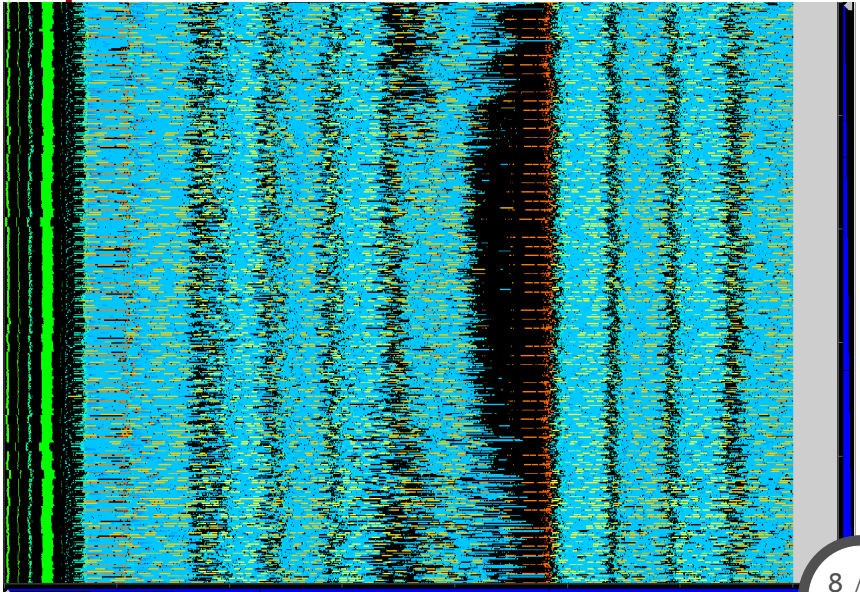
These tools provide details on communication, scaling, mapping, load imbalance, etc.

# Projections

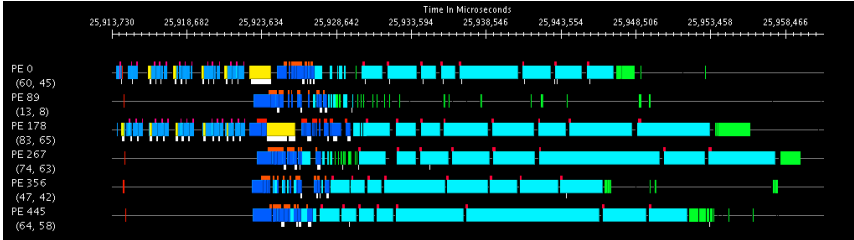
- Projections is a performance analysis tool for Charm++
- Provides tools for analyzing most of the discussed issues in parallel performance
- Runtime traces application executions, creates logs for local analysis



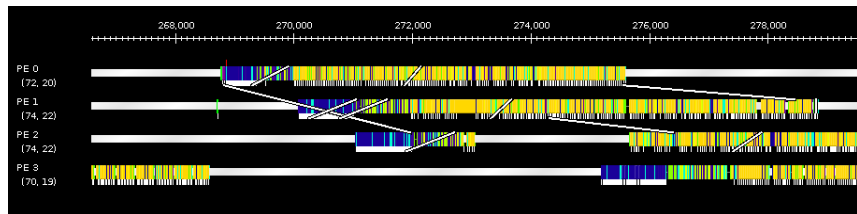
# Projections Tools



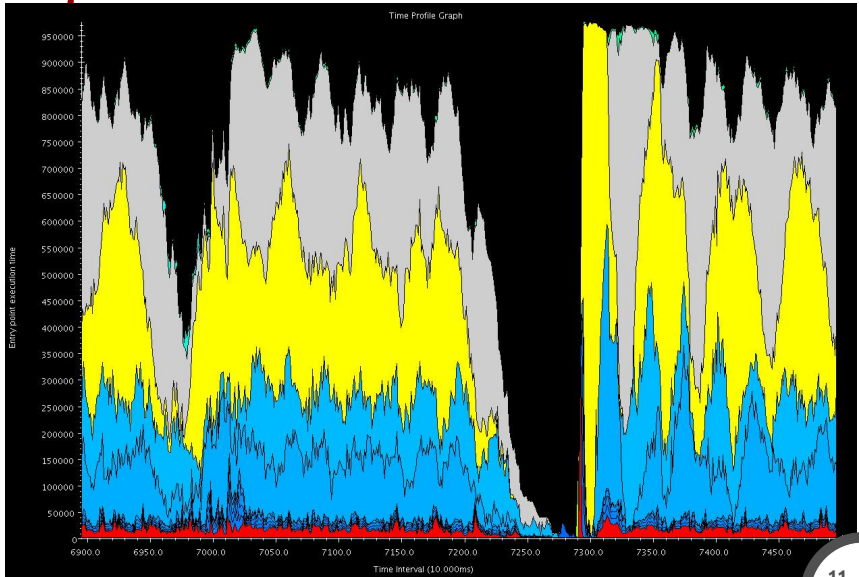
# Projections Tools



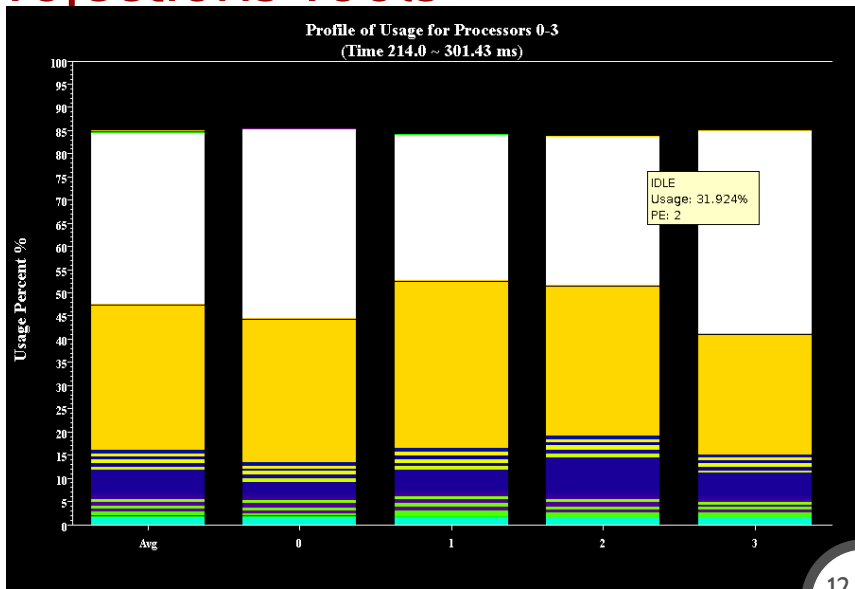
# Projections Tools



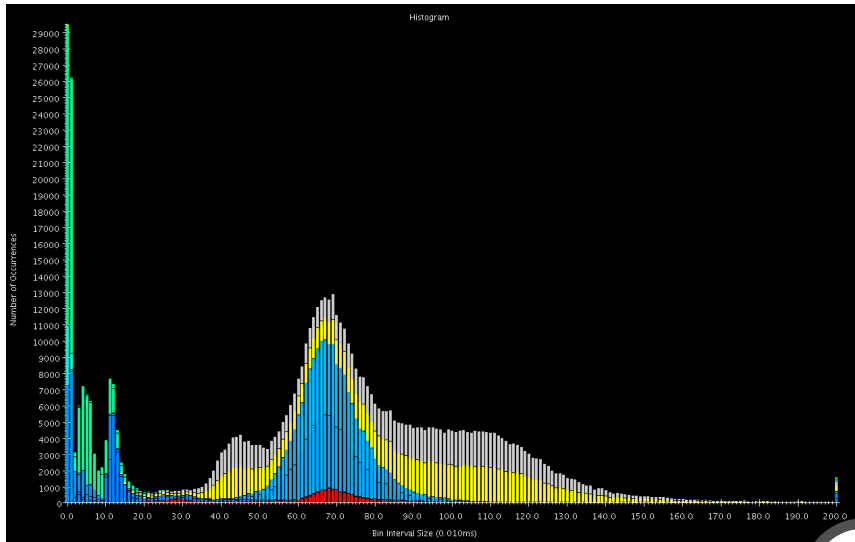
# Projections Tools



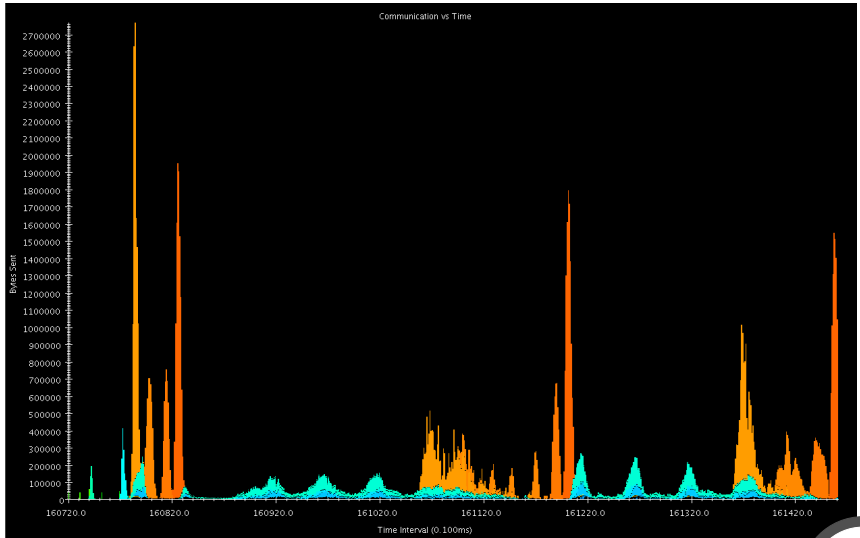
# Projections Tools



# Projections Tools



# Projections Tools



# Recent Advancements in Projections

Recently, we have added some advanced analysis tools to Projections:

- Cache latency measurement
- Communication thread tracing



# Cache Latency Measurement

Using sampling techniques, we have developed a method to trace cache accesses and attribute them to specific lines.

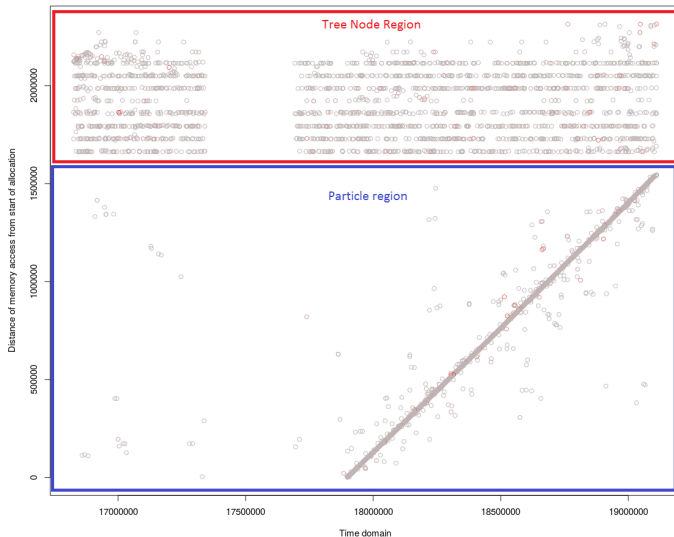
- Intel PEBS - Precise Event Based Sampling
- Provides instruction pointer with sample
- Provides latency of access, not mere hit/miss count

# Cache Latency Measurement

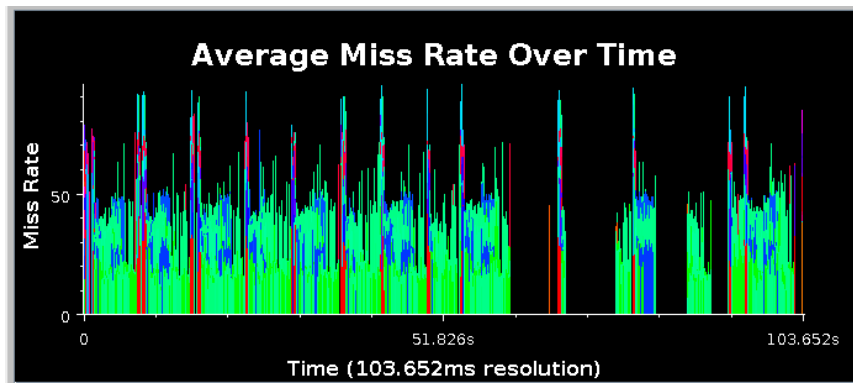
Collected data allows us to analyze:

- Spatiotemporal memory access pattern and latency
- Cache miss rate over time
- Cache latency per source line

# Cache Performance Tools



# Cache Performance Tools



# Cache Performance Tools

The screenshot displays a performance analysis tool with two main windows. The top window shows source code with a conditional block:

```
node->used = true;
#endif
// Always open node if this many particles or fewer.
const int nMinParticleNode = 0;
if(node->particleCount <= nMinParticleNode) {
    return 1;
}

// Note that some of this could be pre-calculated into an "opening radius"
double radius = TreeStuff::opening_geometry_factor * node->moments.radius / theta;
if(radius < node->moments.radius)
```

The bottom window shows a table titled "Memory Access Latency - ././././charmBench/bg5/specfic/changa/ChaNGa.prj.sts". The table has columns for Instruction Pointer, File Name, Line Number, Number of samples, and Source Code. Below the table are radio buttons for "L1 Cache Hits", "L2 Cache Hits", and "LLC cache Hits", with "LLC cache Hits" selected. At the bottom, it says "Update Source Location" and "Select New Range" with "Total samples received: 488111".

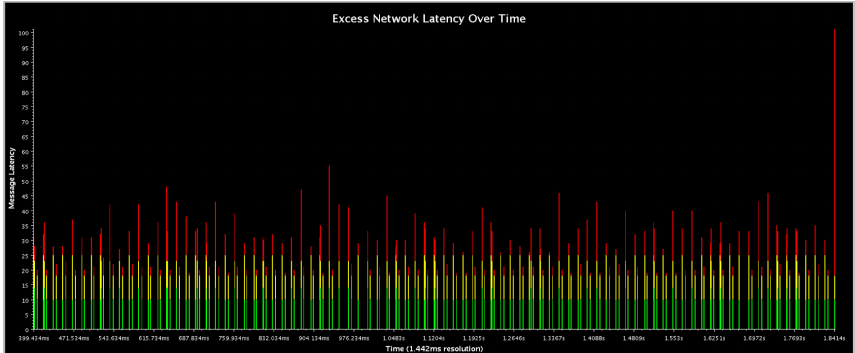
Instruction Pointer	File Name	Line Number	Number of samples	Source Code
0x6307d0	gravity.h	664	89	
0x52fdcf	GenericTreeNode.h	161	59	
0x7f325bc5b0ab		0	55	
0x630804	Vector3D.h	115	49	
0x612fc1		0	46	
0x560ba1	GenericTreeNode.h	360	39	
0x6307dd	gravity.h	669	34	
0x7f325bc5b0c6		0	31	
0x6341bc	Vector3D.h	115	20	
0x634afd	Vector3D.h	115	19	
0x613000		0	13	
0x7f325bc5b6aa		0	12	
0x6130b3		0	11	
0x6341f4	Vector3D.h	115	9	
0x613060		0	9	
0x6130a5		0	8	
0x7f325bc5af82		0	7	
0x613092		0	6	
0x7f325bc003046		0	6	
0x634a82	ParallelGravity.h	1840	5	
0x61321a		0	4	
0x63532e	Compute.C	1427	4	
0x6341d8	Vector3D.h	115	3	
0x62ca5b	TreeWalk.C	345	3	
0x634440	gravity.h	293	3	

# Communication Thread Tracing

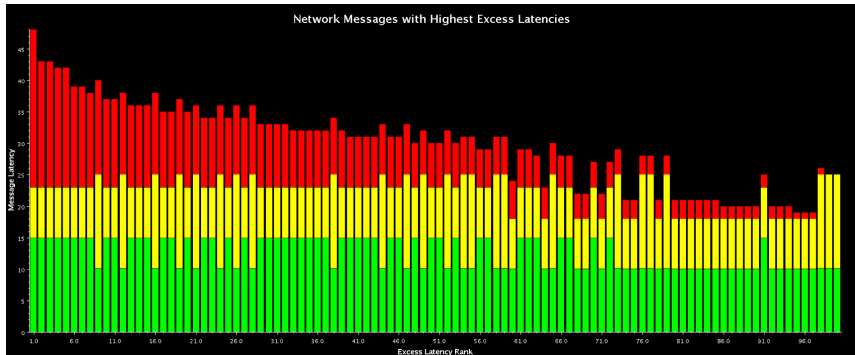
When Charm++ is used in SMP mode, every process has a communication thread. When built with `--enable-tracing-commthread`, the runtime will specifically trace this thread.

Using these logs, we can analyze network performance in detail.

# Communication Thread Tracing

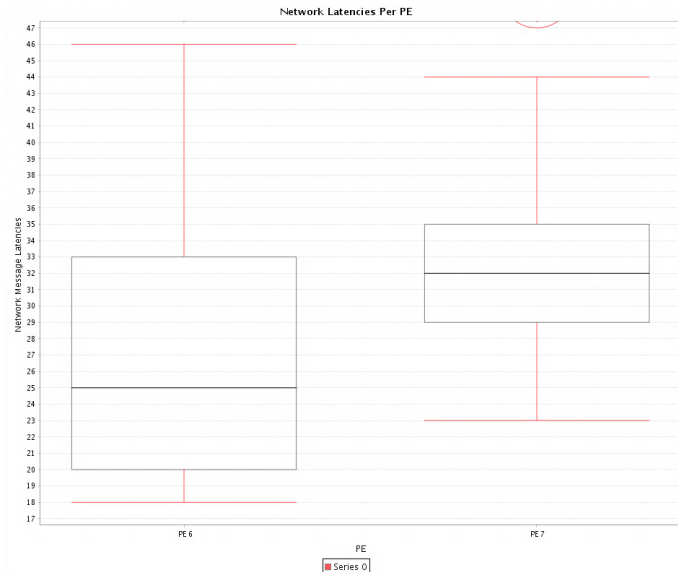


# Communication Thread Tracing





# Communication Thread Tracing



# Communication Thread Tracing

Future work:

- Identify communication thread oversubscription
- Estimate runtime given ideal network
- Use topology and routing information to identify network hotspots

# Conclusion

- Performance analysis tools are *critical* for optimizing HPC applications
- Detailed metrics (performance counters, network delay, etc) can provide insight
- Make performance problems as obvious as possible
- Make it possible for users to write *ad hoc* analysis tools

# Thanks