Capturing Performance Knowledge for Automated Analysis

Kevin A. Huck[1], Oscar Hernandez[2], Van Bui[2],
Sunita Chandrasekaran[3], Barbara Chapman[2], Allen D. Malony[1],
Lois Curfman McInnes[4], Boyana Norris[4]

[1]University of Oregon
[2]University of Houston
[3]Nanyang Technological University
[4]Argonne National Laboratory

SC'08 – Austin, TX – Nov. 20, 2008

Objectives

- To capture and automate performance analysis process and higher level reasoning (meta-analysis)
  - Design flexible analysis components and usable interfaces for integration
  - Engage the parallel programming and tuning environments to use knowledge-based analysis automation capabilities
- Make this available for other problem solving scenarios

Motivation

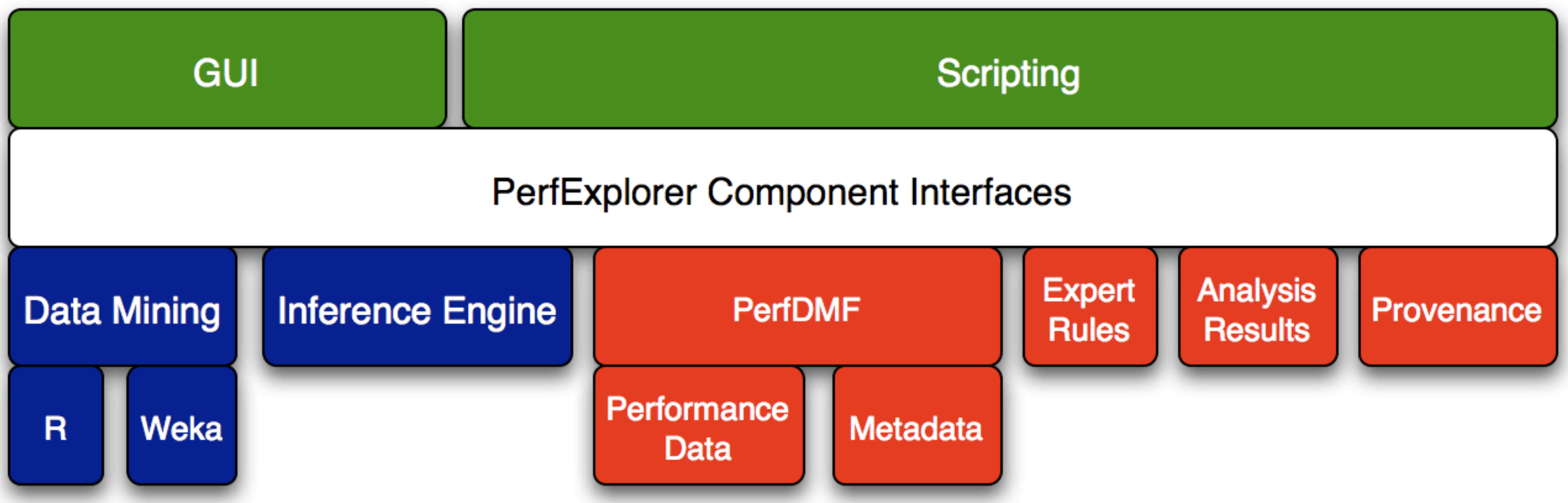
- Parallel performance analysis is complicated and intimidating
  - Management of multi-experiment performance data
  - Application of multi-step processes can introduce errors if done manually
- Lack of support for automation translates to loss of knowledge
  - Which analysis methods are useful for each performance problem type
  - How performance models are obtained and validated
  - How to interpret performance results relative to opportunities for optimization

Application of Analysis Automation

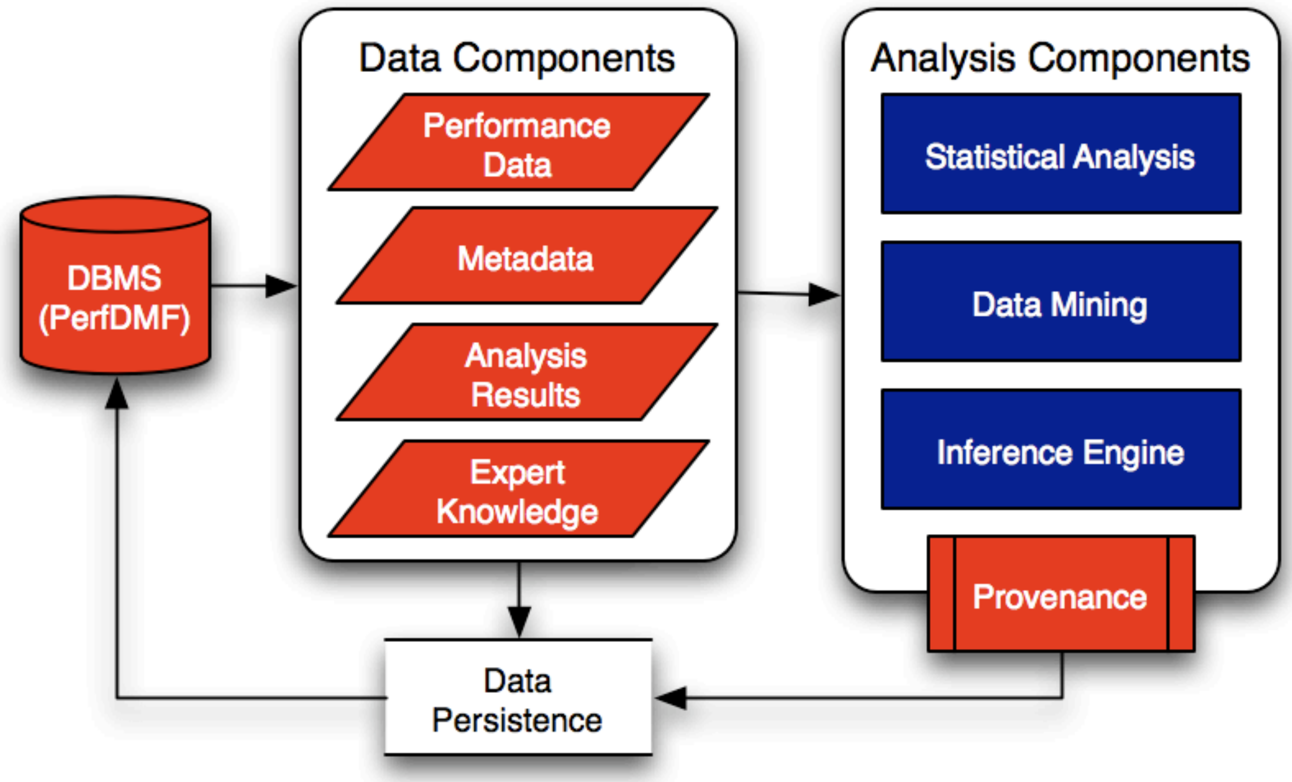
- Application: provide runtime performance data to the OpenUH compiler to improve analysis for optimization (for time, efficiency, power)
- Long term goal: to improve cost model computation for auto-parallelizing code with feedback-based optimization
  - Loop Nest Optimization (LNO)
- Medium term goal: to improve OpenMP performance with feedback-based optimization
- Short term goal: capture expertise from hand-optimized application code as re-usable analysis process

PerfExplorer 2.0

- Data mining framework for parallel profile performance data and metadata
- Programmable, extensible workflow automation
- Rule-based inference for expert system analysis



Automation & Knowledge Engineering



<u>Analysis Components:</u>
Correlation
Derive Metric
Difference
Extractions
K-Means
Smart K-Means
Linear Regression
Log Transform
Merge Trials
PCA
Scale Metric
Split
Process Rules
Save
Draw Chart

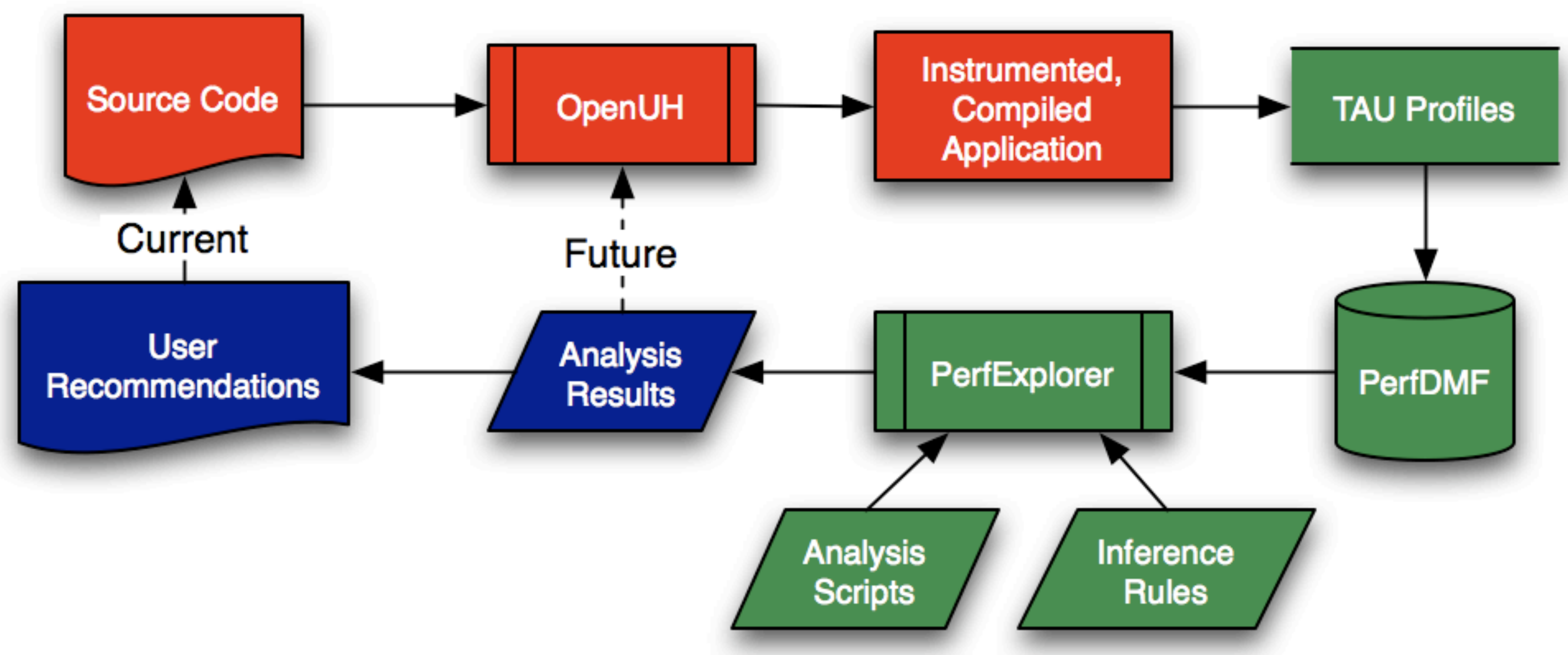
OpenUH Compiler

- C, C++, Fortran95 compiler
- Complete support for OpenMP 2.5
- Front end, IPA and middle/back end:
  - Loop nest optimizer (LNO)
  - Auto parallelizer (with an OpenMP module)
  - Global optimizer (WOPT)
  - Code generator (CG)
- Each module supports feedback-directed optimizations*

OpenUH Cost Model

- Some optimization guided by cost model
  - Loop Nest Optimizer:
    - Processor model
    - Cache model
    - Parallel overhead model
- Cost model computed with static information (and control-flow feedback)
- <u>Long term goal</u>: improve the cost model accuracy using runtime analysis feedback

OpenUH & PerfExplorer Integration



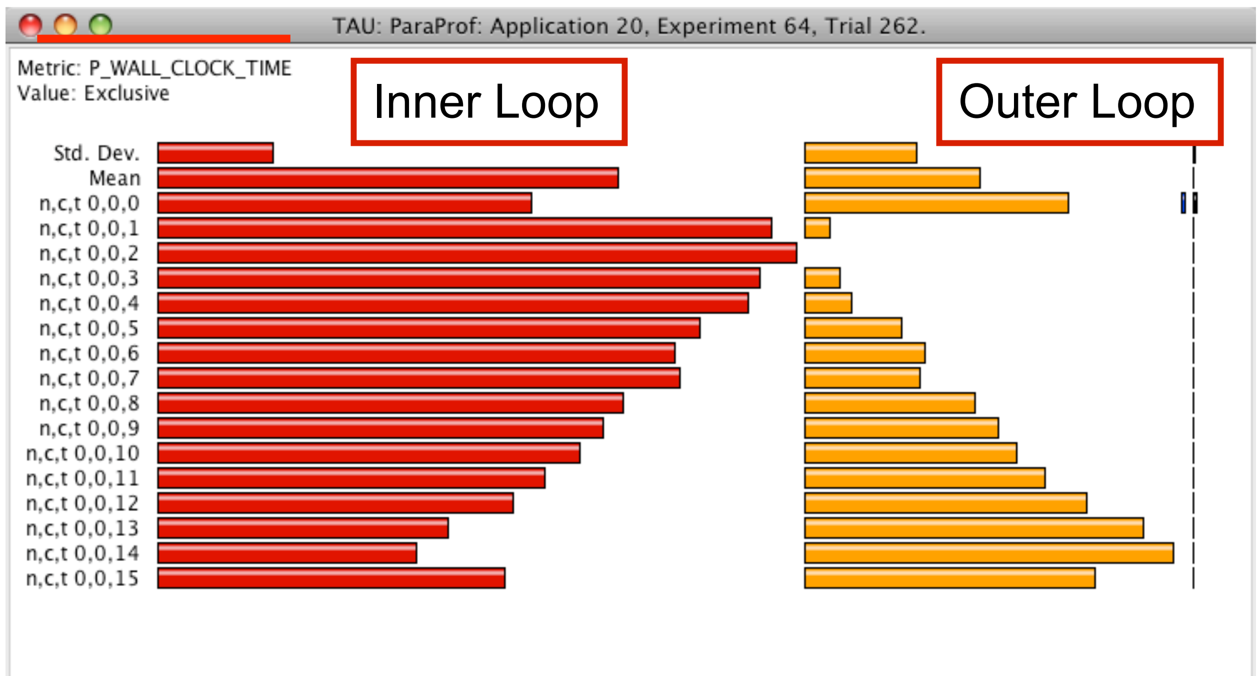
Example #1 – Multiple String Alignment (MSA)

- Compare protein sequences with unknown function to sequences with known function
- Widely used heuristic: progressive alignment (Smith-Waterman)
  - Compute a pairwise distance matrix (90% of time spent here)
  - Construct a guide tree
  - Progressive alignment along the tree
- OpenMP parallelism did not scale well

Photo © NASA

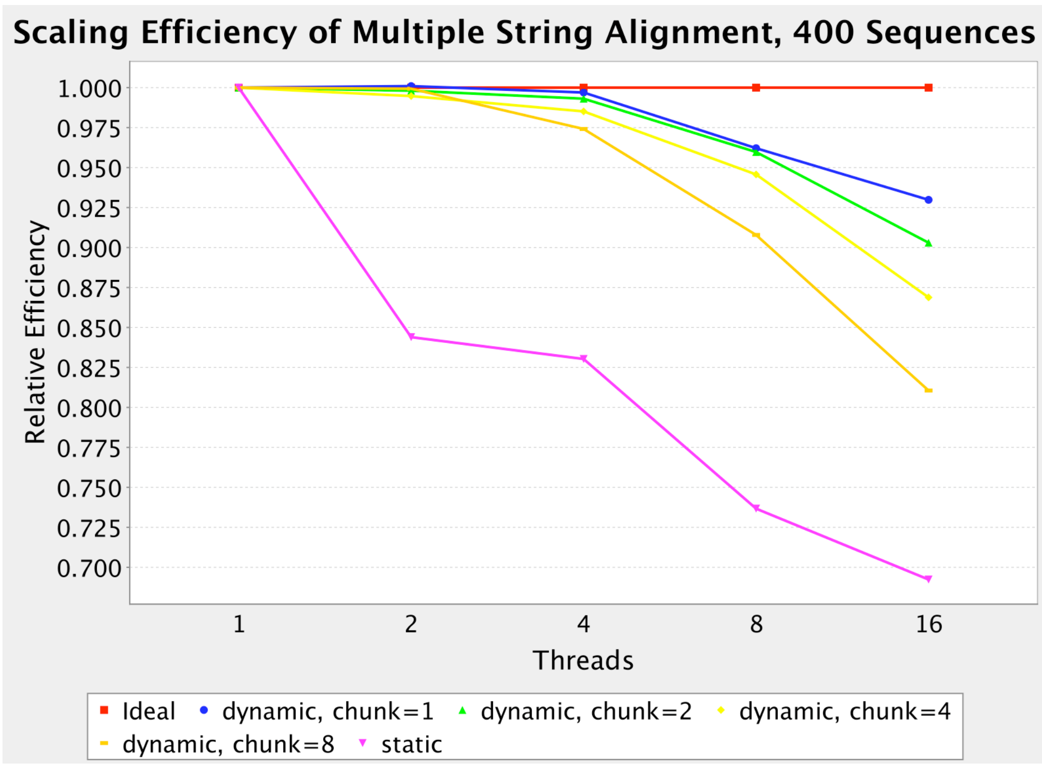
MSA – OpenMP Load Imbalance

```
#pragma omp for
for (m=first; m<=last; m++) {
    for (n=m+1; n<=last; n++) {
        ...
    }
}
```



MSA – Improved Scaling

```
#pragma omp for schedule (dynamic,1) nowait
```



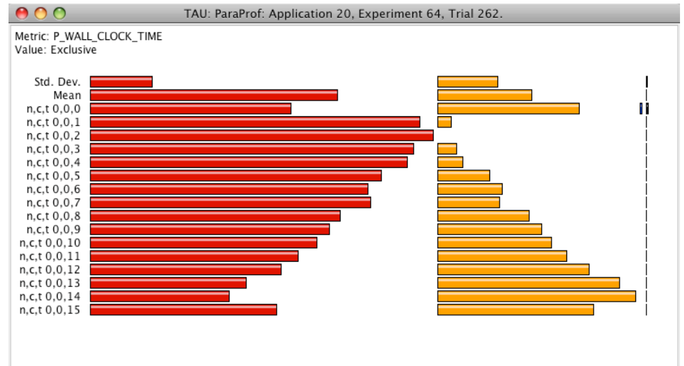
Scheduling parameters

- Before: efficiency < 70% with 16 processors, 400 sequence set
- After: efficiency > 92.5% with 16 processors, 400 sequence set
- Efficiency ~= 80% with 128 processors, 1000 sequence set

Analysis Workflow, Inference Rules

for each instrumented region:

- compute mean, stddev across all threads
- compute, assert stddev/mean ratio
- correlate region against all other regions
- assert correlation
- assert “severity” of event (exclusive time)



Rule1: IF severity(r) > 0.05 AND ratio(r) > 0.25
THEN alert(“load imbalance: r1”) AND assert imbalanced(r)

Rule2: IF imbalanced(r1) AND imbalanced(r2) AND calls (r1,r2) AND correlation(r1,r2) < -0.5
THEN alert(“new schedule suggested: r1, r2”)

Example output

```
--------------- PerfExplorer test script start ------------
--- Looking for load imbalances ---
Loading Rules… Reading rules: openuh/OpenUHRules.drl... done.
loading the data… Main Event: main
Firing rules...

The event LOOP #3 [file:/mnt/netapp/home1/khuck/openuh/src/fpga/msap.c <63, 163>] has a
    high load imbalance for metric P_WALL_CLOCK_TIME
    Mean/Stddev ratio: 0.667, Stddev actual: 6636425.1875
    Percentage of total runtime: 27.15%

The event LOOP #2 [file:/mnt/netapp/home1/khuck/openuh/src/fpga/msap.c <65, 158>] has a
    high load imbalance for metric P_WALL_CLOCK_TIME
    Mean/Stddev ratio: 0.260, Stddev actual: 1.74530281875E7
    Percentage of total runtime: 71.40%

LOOP #3 [file:/mnt/netapp/home1/khuck/openuh/src/fpga/msap.c <63, 163>] calls LOOP #2
    [file:/mnt/netapp/home1/khuck/openuh/src/fpga/msap.c <65, 158>], and they are both
    showing signs of load imbalance.
If these events are in an OpenMP parallel region, consider methods to balance the
    workload, such as dynamic instead of static work assignment.

...done with rules.
---------------- PerfExplorer test script end -------------
```

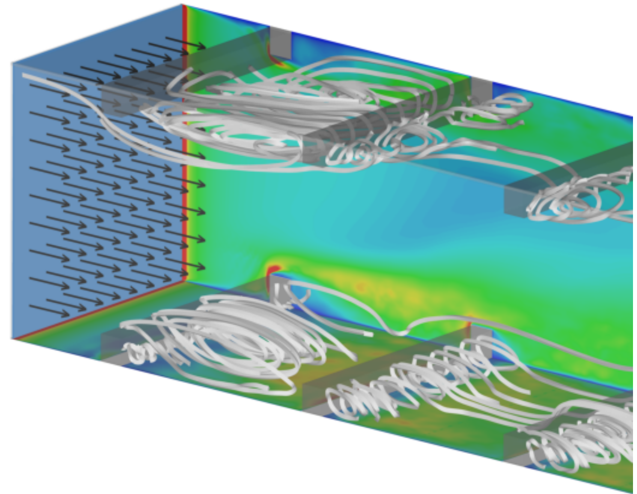
✓ Rule1 true!

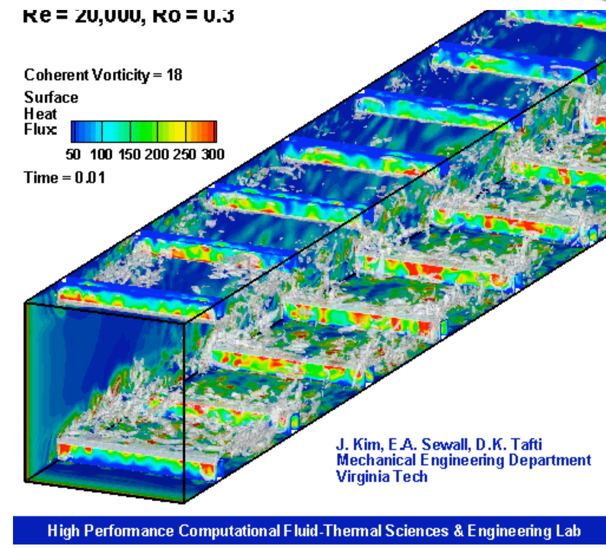
✓ Rule1 true!

✓ Rule2 true!

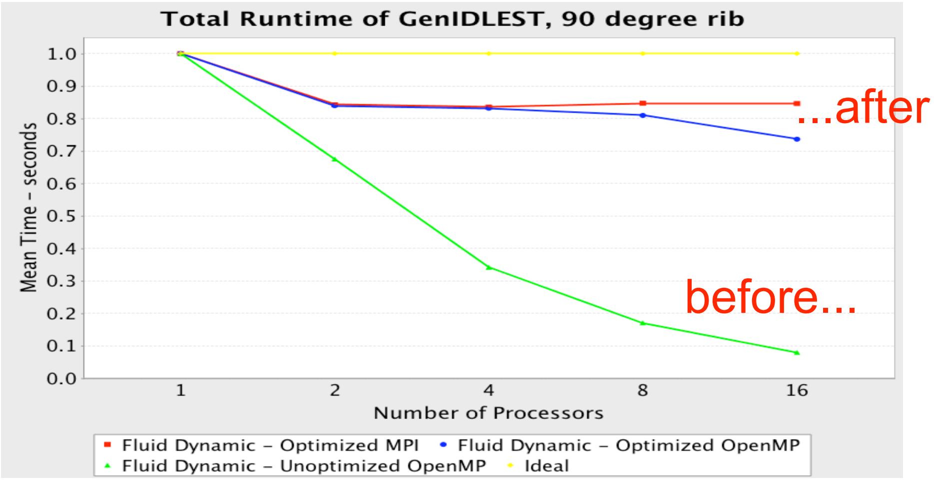
Example #2 – GenIDLEST

- Generalized Incompressible Direct and Large-Eddy Simulations of Turbulence
- Overlapping multi-block body-fitted structured mesh topology, and unstructured inter-block topology
- SPMD parallelism, using MPI and/or OpenMP
- Test cases: investigate turbine cooling duct, 45 and 90 degree ribs
  - Detached Eddy Simulations (45)
  - Large Eddy Simulations (90)

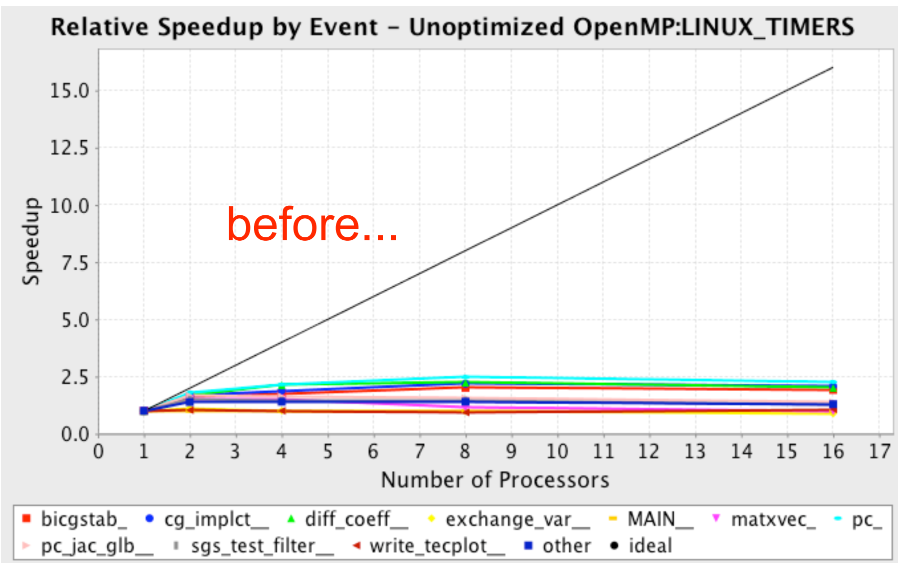


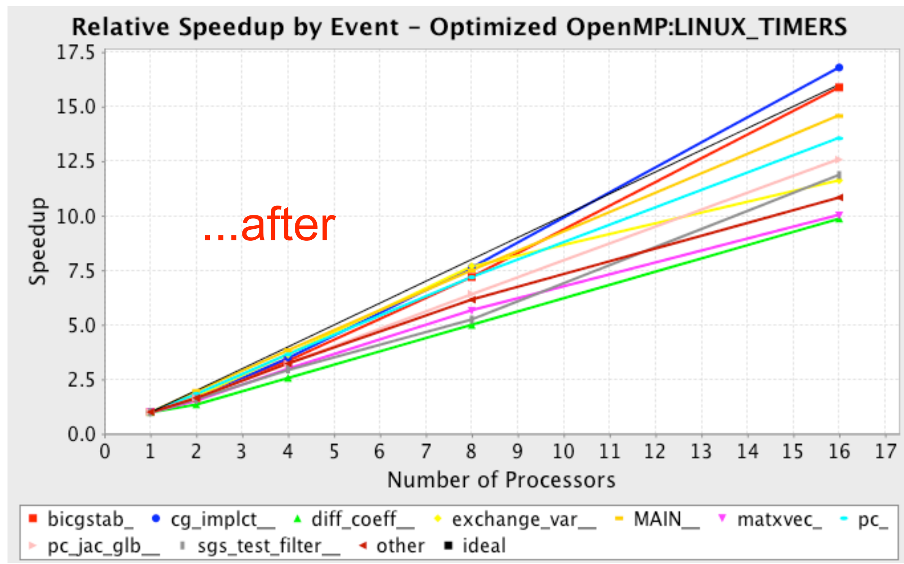


GenIDLEST OpenMP Scaling



Problems mainly related to remote memory references on NUMA architecture, excessive memory copies initiated by master thread





Analysis Workflow, Inference Rules

for each instrumented region, exclusive:
- derive, assert inefficiency metric
- derive, assert memory/total stall cycles metric
- derive, assert memory cycles metric
- derive, assert remote memory accesses ratio metric
- assert “severity” of event

also compute values for main, inclusive

Rule1: IF severity(r) > 0.02 AND inefficiency(r) > inefficiency(main)
THEN alert (“inefficient, r”) AND assert(inefficient(r))

Rule2: IF inefficient(r) AND tsm(r) > 0.9
THEN alert (“memory stalls, r”) AND assert (memstall(r))

Rule3: IF memstall(r) AND memory(r) > memory(main)
THEN alert (“memory cycles, r”)

Rule4: IF memstall(r) AND remote(r) > remote(main)
THEN alert (“remote references, r”)

Example output

```
Firing rules...

The event exchange_var__ has a higher than average stall / cycle rate
   Average stalls per cycle: 0.79877, Event stalls per cycle: 0.95439
   Percentage of total runtime: 31.16%
...
The event exchange_var__ has a high percentage of stalls due to L1 data
   cache misses and FP Stalls.
   Percent of Stalls due to these two reasons: 99.88%
...
The event exchange_var__ has a higher than average number of cycles
   handling memory references.
   Average memory cycles: 73.72%, Event memory cycles: 100.09%
...
The event bicgstab_ has a lower than average local memory reference
   percentage.  If this is an OpenMP parallel region, consider methods for
   parallelizing data initialization.
   Average percentage: 93.77%, Event ratio: 90.44%

...done with rules.
---------------- JPython test script end -------------
```

✓ Rule1 true!

✓ Rule2 true!

✓ Rule3 true!

✓ Rule4 true!

Example #3 – Power Estimation

- May want to optimize for metric other than time
- Hardware counter data can be used to estimate power consumption
- Simplified model – Itanium2:

scaling factor   max power

CPU = (instructions / cycles) * 0.0459 * 122

L1 = (L1 references / cycles) * 0.0017 * 122

L2 = (L2 references / cycles) * 0.0171 * 122

L3 = (L3 references / cycles) * 0.935 * 122

TOTAL = CPU + L1 + L2 + L3

Power Estimation – Results

Metric	-O0	-O1	-O2	-O3
Time	1.0	0.338	0.071	0.049
Instructions Completed	1.0	0.471	0.059	0.056
Instructions Issued	1.0	0.472	0.063	0.061
Instructions Completed Per Cycle	1.0	1.397	0.857	1.209
Instructions Issued Per Cycle	1.0	1.400	0.909	1.316
Power Consumed (Watts)	1.0	1.025	1.001	1.029
Energy Consumed (Joules)	1.0	0.346	0.071	0.050
FLOP/Joule	1.0	2.867	13.684	19.305

Future Work

- Modify cost model calculation to integrate feedback from runtime data analysis
- Feed information about sources of overhead and causes to OpenMP infrastructure
- Implement strategies for variable privatization and first touch policies
- Parallel model could be improved for auto-parallelized code
- Optimizations for performance and power

Conclusion

- Initial work into capturing analysis process
- Automation and expert knowledge to direct processing, interpret results, and provide decision support
- Flexible scripting, rule-based system is reusable, extensible to other analysis scenarios

Acknowledgements

- US Department of Energy (DOE)
  - Office of Science
- US National Science Foundation (NSF)
- Argonne National Lab
- NASA / CSC (Altix 300)
- NCSA (Altix 4700)
- Virginia Tech (GenIDLEST application)