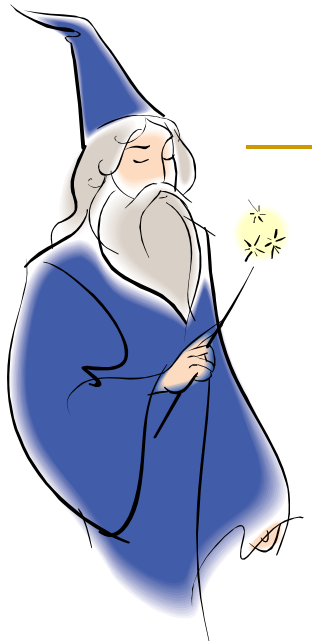
UF UNIVERSITY of FLORIDA

www.hcs.ufl.edu
High-performance Computing & Simulation Research Lab

Parallel Performance Wizard: An Infrastructure and Tool for Analysis of Parallel Application Performance



Presenter: Hung-Hsun Su

Advisor: Dr. Alan D. George

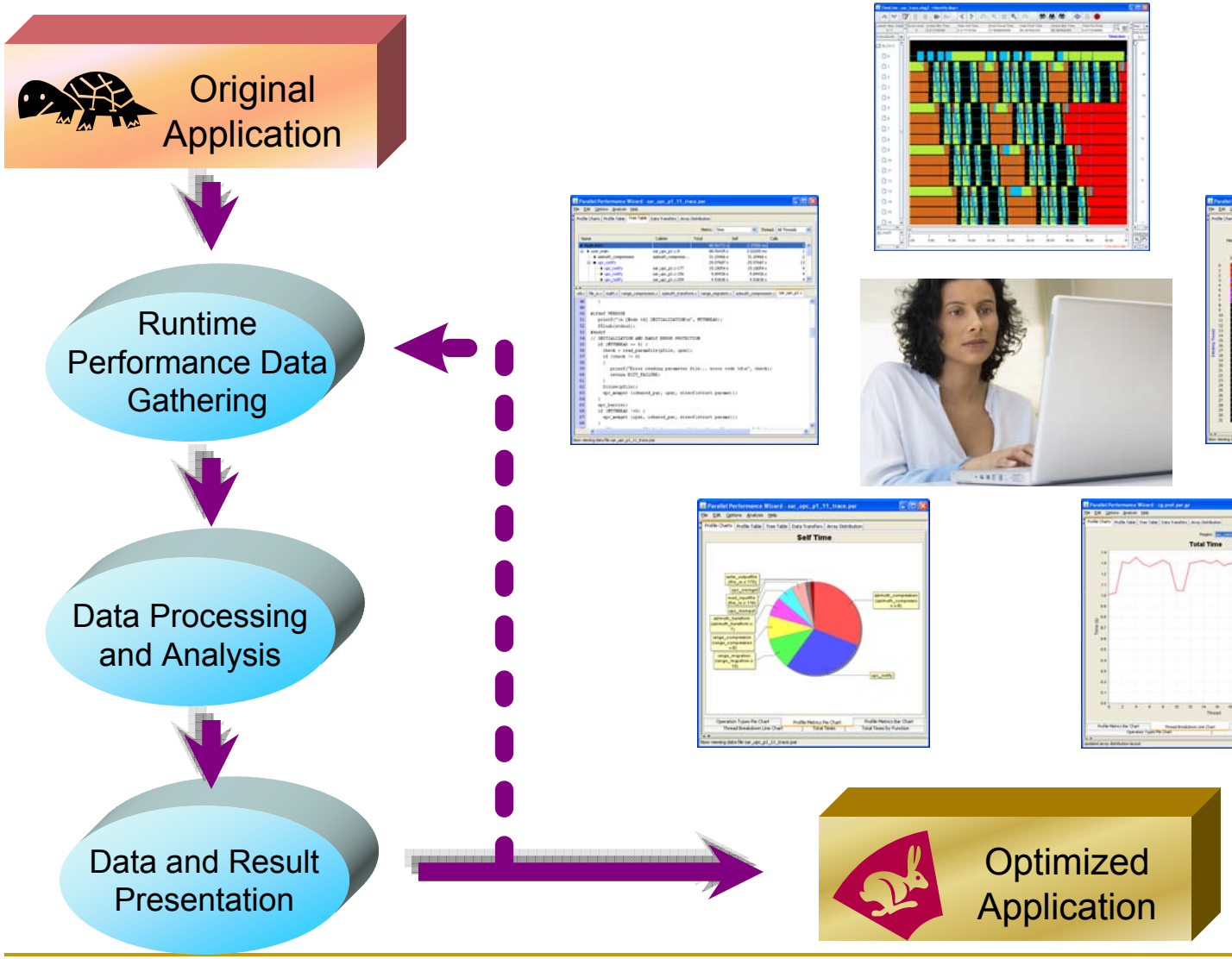
Electrical and Computer Engineering Dept.,
University of Florida

Need for Parallel Performance Analysis Tool

- Computationally intensive parallel applications are constantly being developed in many scientific fields
- Parallel programming models supply application writers with means to express parallelism in a given environment
- Unfortunately, the added complexity of the environment and model makes it more difficult to optimize the application to achieve a desirable performance level

- Performance Analysis Tools (PATs) *increase productivity* by making application optimization process simpler for user

Role of Performance Analysis Tool



Need for Generalized Tool Infrastructure

- Development of a performance analysis tool is a time-consuming process (takes years to develop)
- Quite a few performance analysis tools exist

- However, majority of them support *Message Passing Interface (MPI)* with very few supporting other models such as those in the *Partitioned Global Address Space (PGAS)* family

- One of the reasons for the limited model support is that these tools were designed and developed specifically to target a single model
  - Tool is too tightly coupled with the original model, making it cumbersome to add new model support
- A generalized performance analysis tool infrastructure would help in this aspect

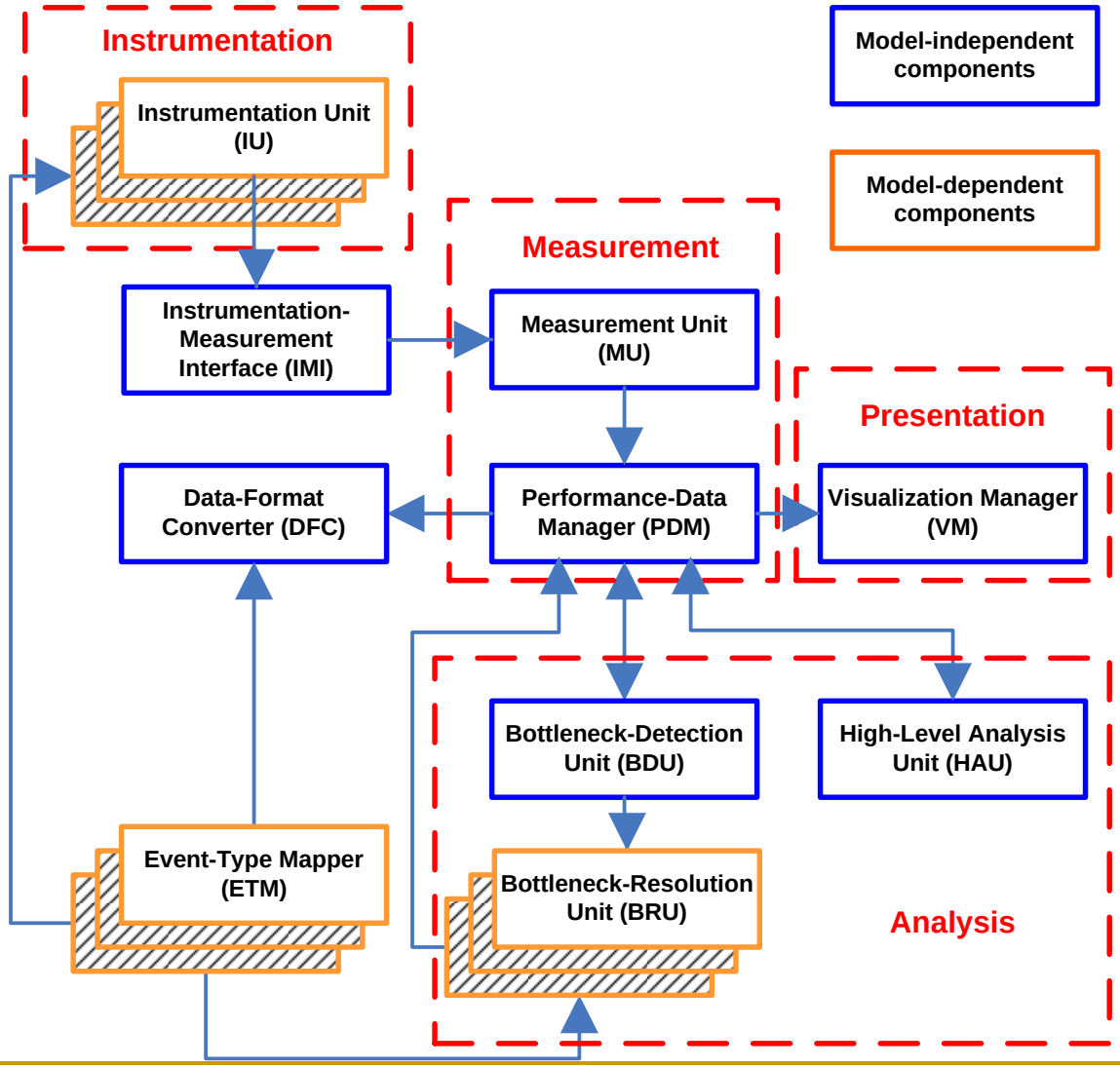
Properties of a Generalized Infrastructure

- Uses a generic operation type abstraction
  - Each model construct is mapped to a generic operation type
  - Tool is designed to work largely with generic operation types
    - Components that only use generic operation types are model-independent (i.e. reusable across models)

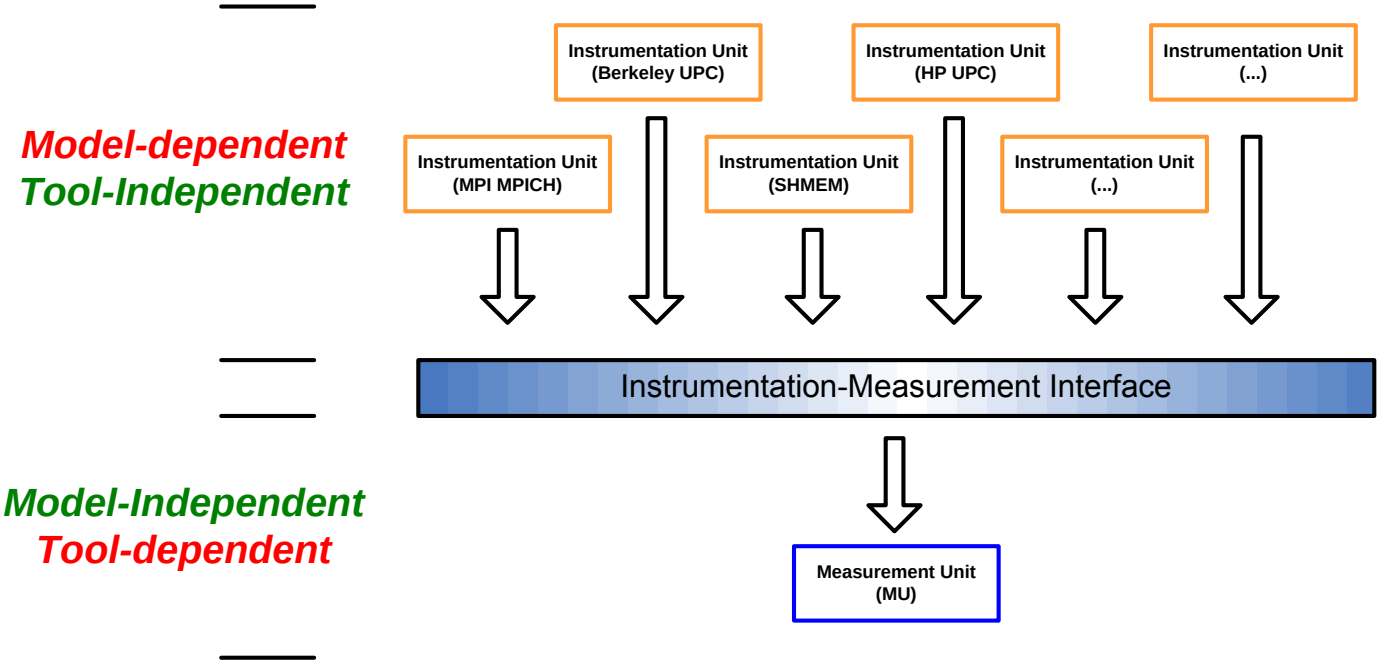
- The goal is to minimize the number of model-dependent components

Data exchange (P2P)	***Pair-wise synchronization***	***Group-wise synchronization***	***Local processing***
One-sided (put, get, fence)	Lock manipulation	Barrier	Work distribution (for-all)
Two-sided (send, receive, wait)	Wait on remote (spin lock, atomic swap, etc.)	Collectives	User functions & I/O operations

PPW High-level Framework



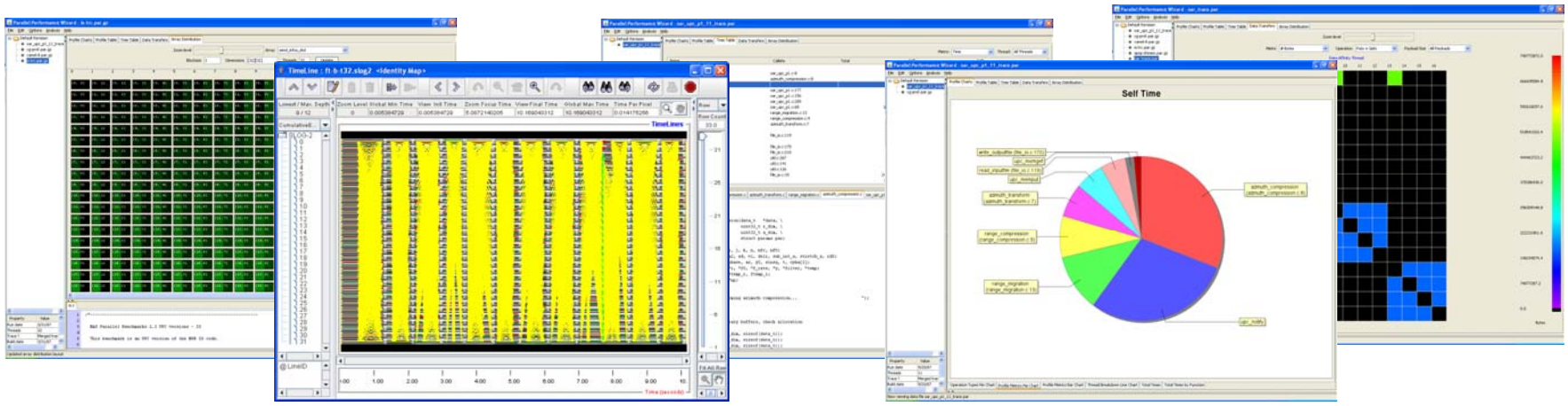
Instrumentation-Measurement Interface (GASP)



- Different instrumentation techniques (adding code to collect performance data) are applicable to different programming model implementations
- However, one generic measurement unit is sufficient to record data
- Standardized instrumentation-measurement interface (a.k.a. GASP) facilitates the transition from multiple instrumentation units to a single measurement unit

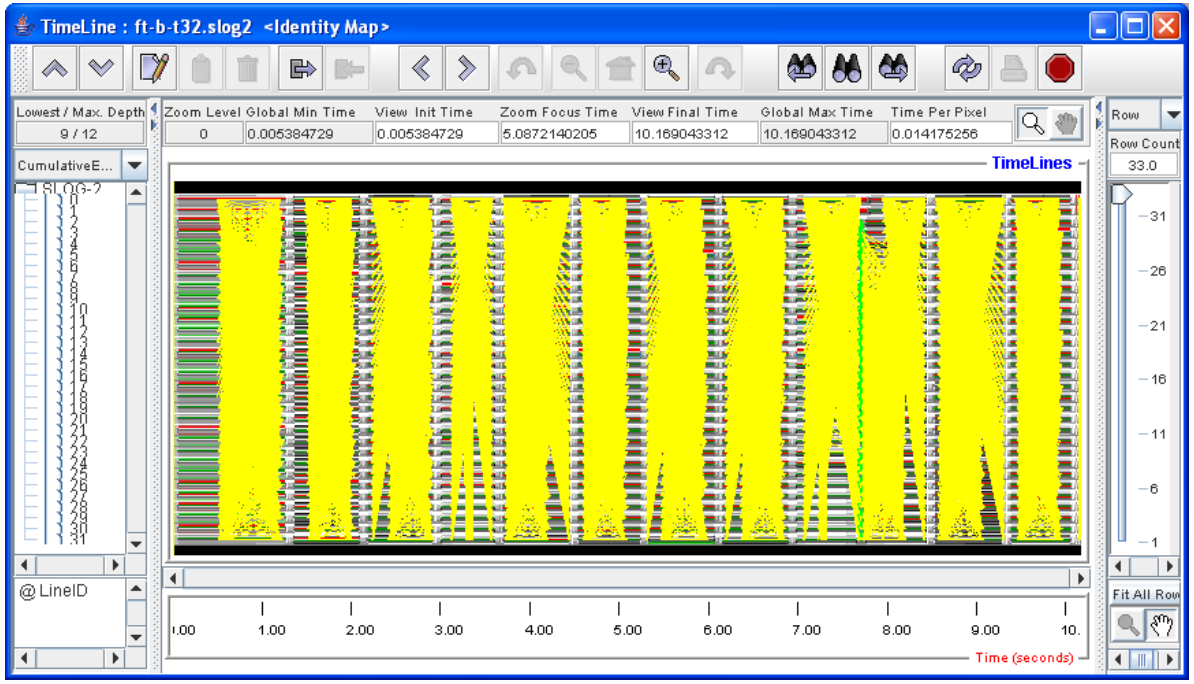
PPW Model Support

- PPW infrastructure was first implemented to support Berkeley UPC
  - Took approximately 1-2 years to develop (sans bottleneck detection)
  - Supports one-sided transfer, global synchronization, locks, etc. operation types
- Quadrics SHMEM and MPICH MPI were then quickly added
  - Took about 3-6 month to complete
  - Instrumentation provided via PSHMEM/PMPI interface with calls to GASP
  - Majority of components remained unchanged
    - Minor modification made to measurement unit and visualization unit to support collectives and two-sided transfers



Why do Automatic Analysis of Data?

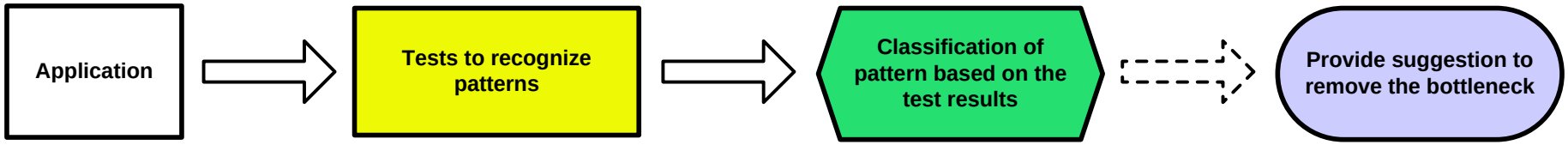
- With a long running and/or complex application, the amount of performance data available can be overwhelming
- Automatic performance analysis helps by presenting a set of useful information out of a much larger data set



Automatic Performance Analysis

- Pattern is a description of a program behavior
  - Expert programmers know how to recognize performance patterns where an everyday programmer may not
  - Fair amount of performance patterns generated over the years by researchers

- Automatic performance analysis of an application
  - Performs a series of tests to recognize patterns
  - Classify the pattern base on the result of tests
  - Suggest possible solution to remove the bottleneck (to some extent)



Pattern Categorization

- Patterns are categorized into one of the following three levels
  - Experiment set level
    - Compare performance for a set of experiments
    - Pattern example: poor scalability of code

$\frac{Time(p) \times p}{Time(q) \times q}$	Ratio < 1; non-Ideal application speedup of application/region	Algorithmic change for the application/region

  - Application level
    - Provide an overview of overall application performance
    - Pattern example: lots of small data transfers

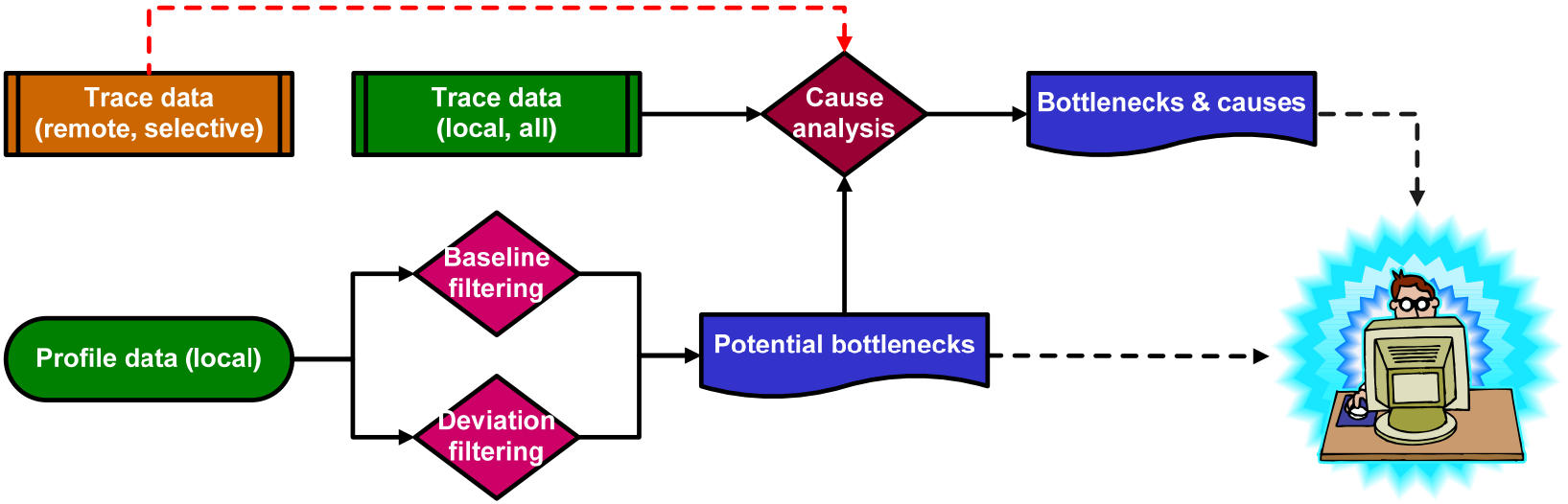
$\frac{Count(data.transfer)}{Time(transfer)}$	Ratio >= THRESHOLD; lots of small data transfers	Aggregate data transfers

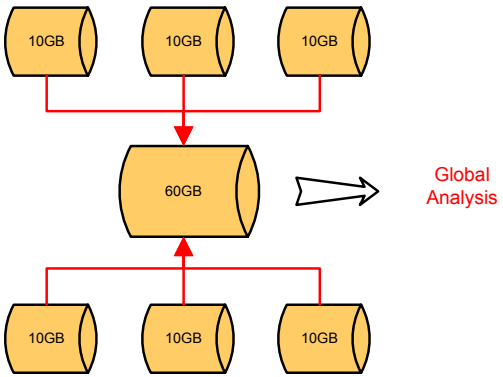
  - Node level
    - Enable detailed analysis of performance data that helps to pinpoint the exact location and cause of the bottleneck
    - Pattern example: 2nd invocation of send on node 2, line 10 is a late sender

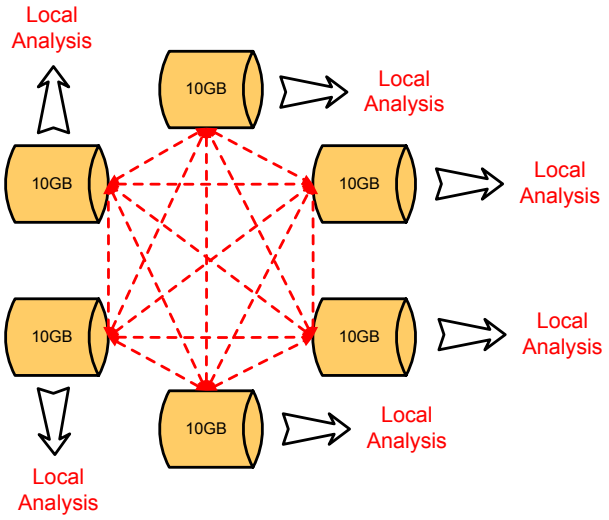
Node Level Patterns

- Each node performs independent analysis using local data and a small amount of data from other node when needed
  - Each node tries to minimize its execution time
  - Observed application execution time ≈ execution time of the longest running node
- Pattern tests aim to detect deviation from the optimal situation
  - Excessiveness analysis: large number of operation occurrences
    - Frequency evaluation (lots of operations in a short time?)
    - Excessive operation evaluation (operation could be eliminated?)
  - Delay analysis: long running operations
    - Baseline approach (actual time >> expected?)
    - Variant approach (min_time << avg_time *or* max_time >> avg_time?)
- Patterns are defined in term of the generic operation types, thus applicable to any model

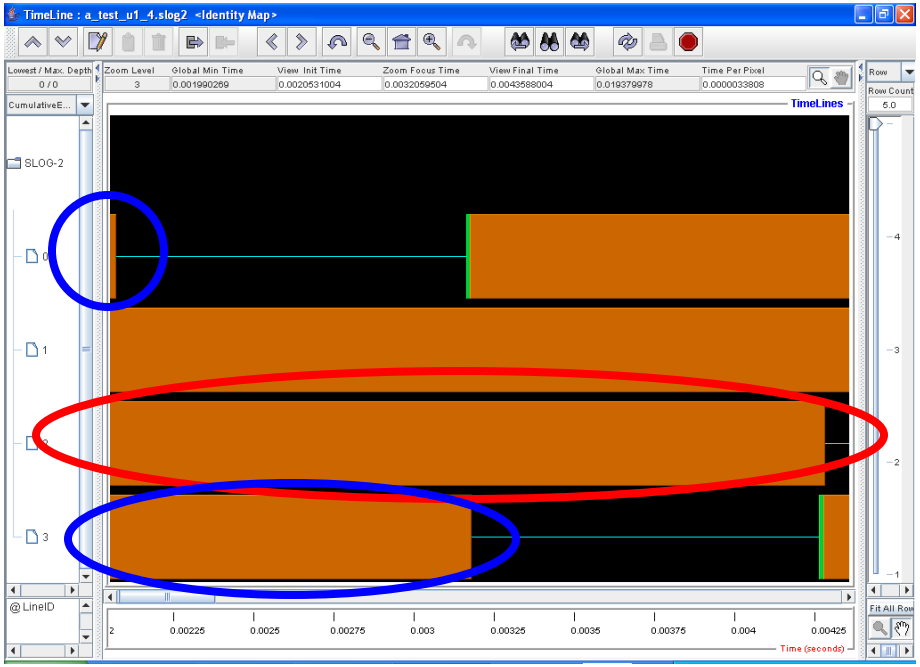
Node Level Detection Mechanism







Example Node-Level Pattern: Lock Delay



Lock (x) at line 2, Time: 10 → 100

↓

Send "Lock (x, 10, 100)" to all other nodes

↓

Receive from Node 0
Unlock(x), end (line 3, instance 1) = 26

Receive from Node 3
Unlock(x), end (line 3, instance 1) = 56

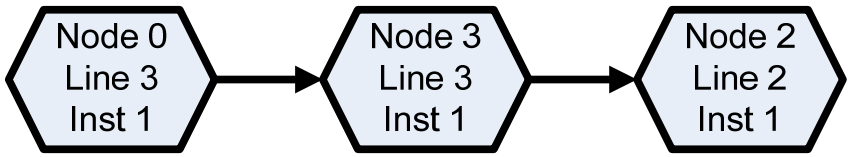
Node 2's Potential Bottleneck List

…
Locks
Lock at Node 2, Line 2

A-2-A
…

Node 2's Local Trace Records

…
Lock (x) at line 2, Time: 10 → 100
…
Lock (x) at line 5, Time: 100 → 120
…
Lock (x) at line 2, Time: 120 → 140
…



Example Analysis Result: Lock Delay

Total Time	= 7.75E07 ns
Computation Time	= 2.51E07 ns, Ratio = 32.45%
Communication Time	= 0.00E00 ns, Ratio = 0.00%
	Count = 0.00E00,
	Bandwidth = 0.00 MB/s [low bandwidth means lots of small data transfers]
Global Sync Time	= 9.52E06 ns, Ratio = 12.28%
P2P Sync Time	= 4.28E07 ns, Ratio = 55.27%
Comm / Comp Ratio	= 0.0000 [Low number means more work done]
Sync / Comp Ratio	= 2.0821 [Low number means low overhead]
Btnk Time	= 5.22E07 ns, Ratio = 67.36% [of program with performance bottlenecks]

Ratios showing that lots of time were lost due to data transfer and synchronization

XXXXXXXXXX BOTTLENECKS (S) XXXXXXXXXX
Found total of 7 filtered bottleneck(s)
…
…

--- P2P Operations (S) ---
…
Node#2, Line#2, upc_lock(UPC), T(avg) = 2.40E6, T(exp) = 5907, 5 Call(s), Ratio = 4.06E2, 15.45%
Program, 15.42% Degradation
  |---> [Instance#1] Ratio = 3.78E2, 0.0020 s --> 0.0043 s, Duration = 0.0022
      |--> [Node#0-Line#3] :Wait for lock to be avaliable from specified node;
      |--> [Node#3-Line#3] :Wait for lock to be avaliable from specified node;
…
…
XXXXXXXXXX BOTTLENECKS (E) XXXXXXXXXX

At node 2, upc_lock at line 2 (1st occurrence) executed much slower than expected

More detailed analysis reveals node 2 waits on node 0 and 3 to release lock

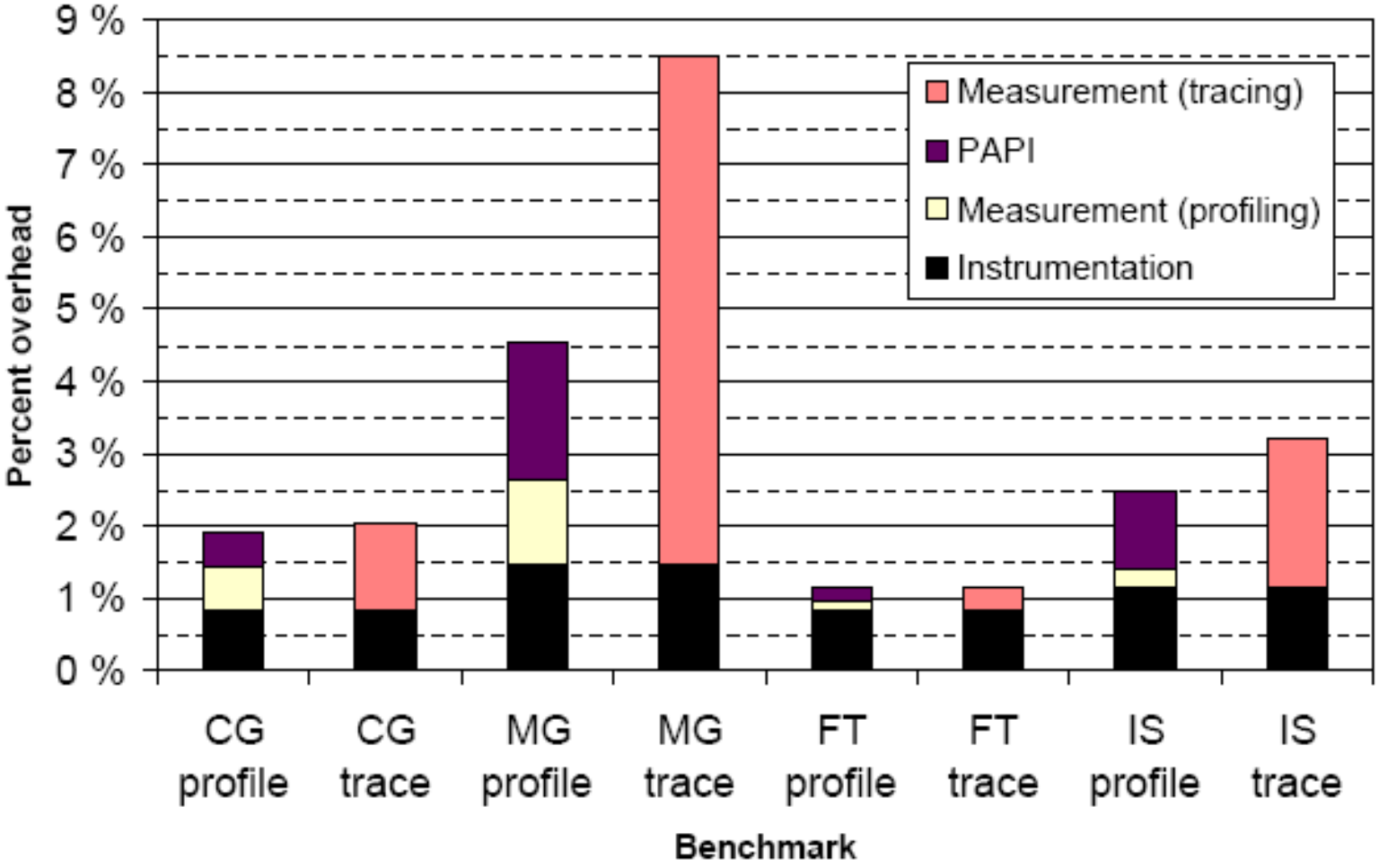
Conclusions

- Parallel Performance Wizard is an infrastructure designed to support multiple parallel programming models with ease
  - Uses the generic operation type abstraction that improves the re-usability of the system components
- A new automatic performance analysis approach is currently being developed and tested
  - Captures known performance patterns in one of the three levels
  - Employs a distributed detection method to improve execution time and minimize data transfer among nodes
  - Potential to support multi-model and multi-level analysis
- A working implementation of PPW is now available for UPC, SHMEM and MPI
  - For more information see http://ppw.hcs.ufl.edu

Acknowledgements

- Department of Defense
  - Funding the PPW project
- Dr. Alan D. George
  - Advisor for my research
- Adam Leko, Max Billingsley III, Bryan Golden, Hans Sherburne [U. of Florida]
  - Design discussion of infrastructure, low-level design and implementation, visualization generation
- Dan Bonachea [U.C. Berkeley]
  - Berkeley GASP discussion and implementation

Supplement Slide 1 - Instrumentation-Measurement Interface Overhead



Profile Filtering Performance Improvement