Performance Measurement of Novice HPC Programmers' Code

Rola Alameh

Nico Zazworka

Jeffrey K. Hollingsworth

Presented by Rola Alameh

Outline

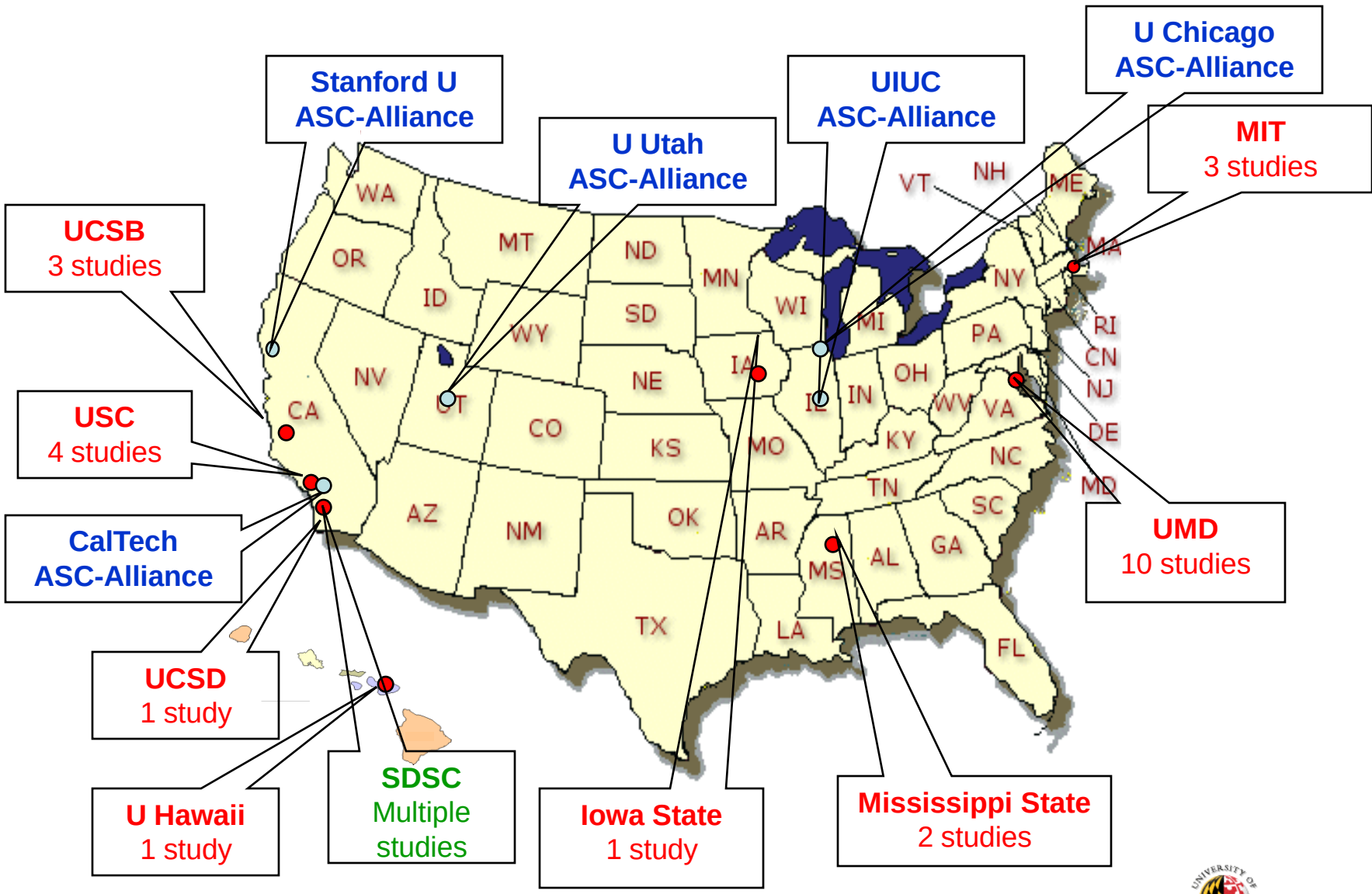
- Why study novice programmer's code performance?
- How to study performance? APMS
- Parameter Specification problem
- Preliminary studies

DARPA High Productivity Computing Systems

- Problem
  - Build sufficient knowledge about HPC to improve the time and cost of developing these codes
- Project Goal
  - Improve the buyers ability to select
    - the high end computer
    - for the problems to be solved
    - based upon productivity

  Time to Solution = Development Time + Execution Time
- Research Goal
  - Develop theories, hypotheses, and guidelines that allow us to characterize, evaluate, predict and improve how an HPC environment (hardware, software, human) affects the development of high end computing codes.
- Partners: MIT Lincoln Labs, MIT, UCSD, UCSB, UMD, USC, FC-MD

Study Locations



Areas of Studies

- Effort
  - OpenMP saves 35-75% of effort vs. MPI on most problems
  - Experience with problem reduces effort, but effect of programming model is greater than effect of experience
- Defects
  - Defect Classification Scheme
  - Defect “Experience Base”: http://www.hpcbugbase.org
- Process flow
  - What is the normal process followed?
  - What is the breakdown between work and rework?
  - Can we discover the process steps from lo level automatically collected data?

Performance Studies

- From a software engineering perspective
  - Relating performance to user and context variables
    - NOT machine and architectural parameters
  - Is there correlation between effort and performance?
  - When performance is the goal:
    - do experts and students spend the same amount of time?
    - do experts get significantly better performance?
- In this presentation
  - First look at these questions
  - Two class assignments:
    - conjugate gradient
    - game of life
  - Programmed using MPI + C

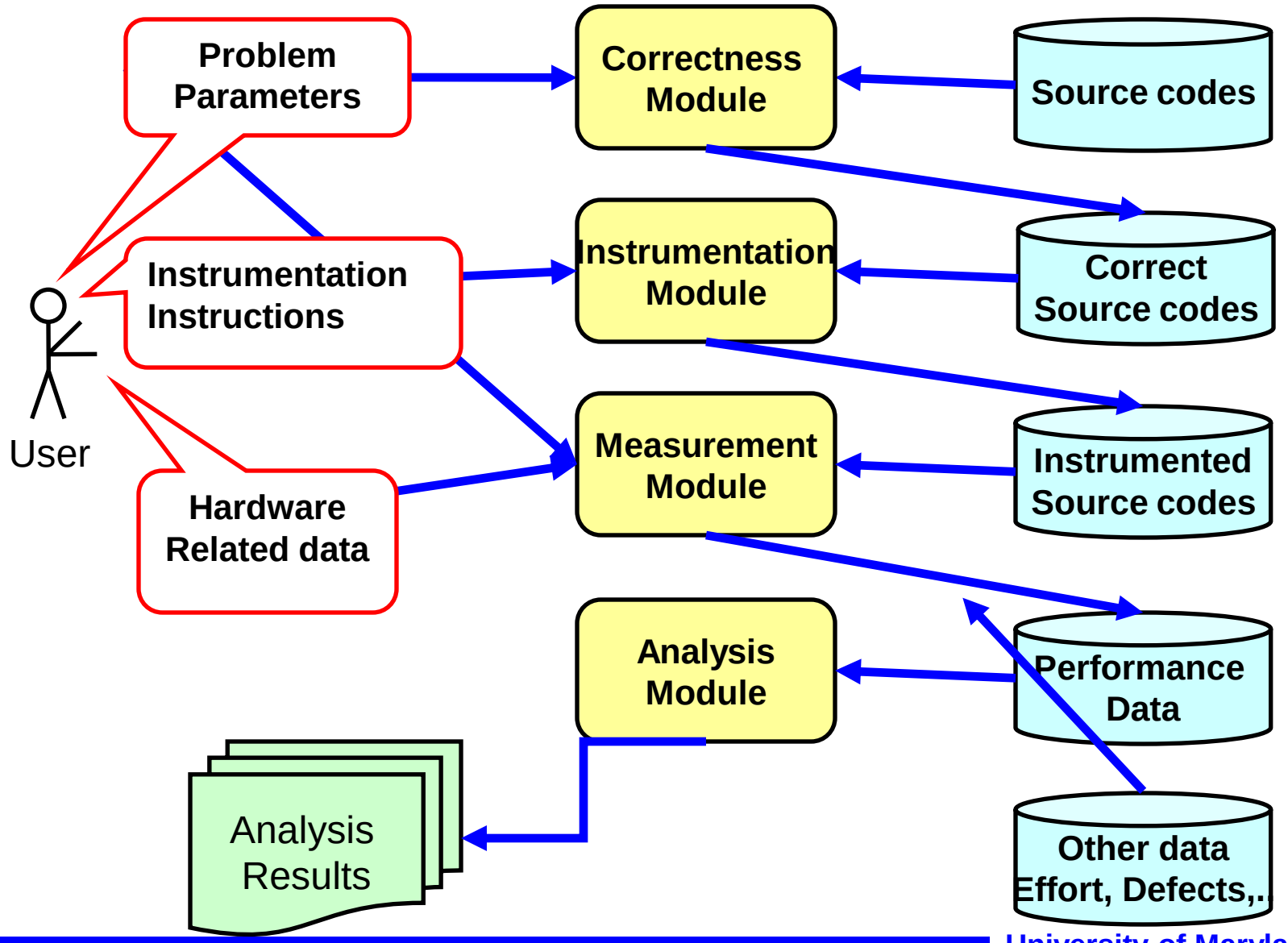
Performance Study challenges

- Need many runs
  - multiple versions
  - many students
  - different input sets
  - vary number of processors
- Similar assignments
  - differ in input and output formats
  - have differences even within the same class
- Our solution: Automated Performance Measurement System

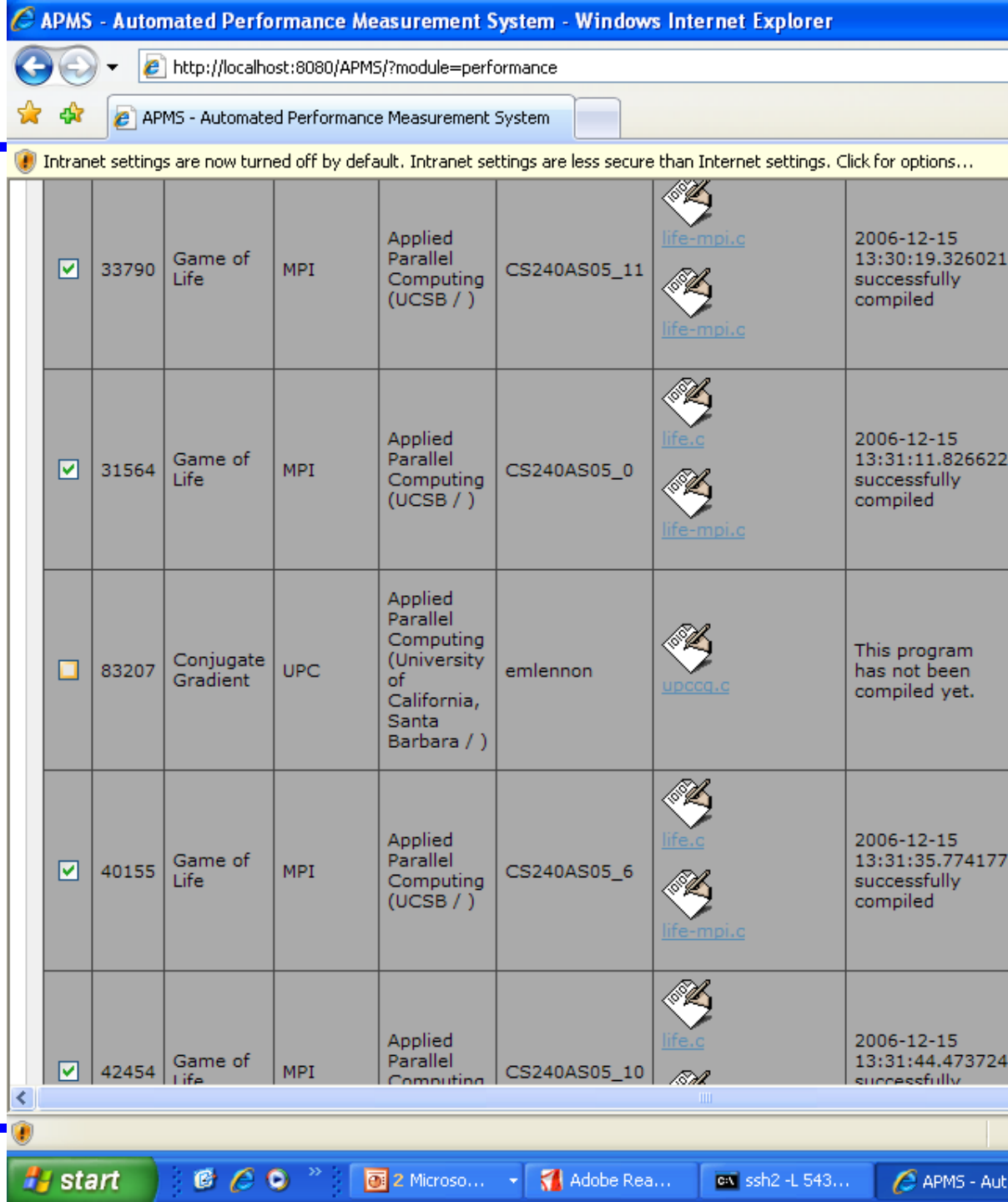
APMS: Features

- Supports multiple
  - parallel programs
  - sets of input parameters
  - runs of the same program
- Records
  - program output
    - displays it for judging program correctness
  - timing data
    - used to judge performance

The System Design

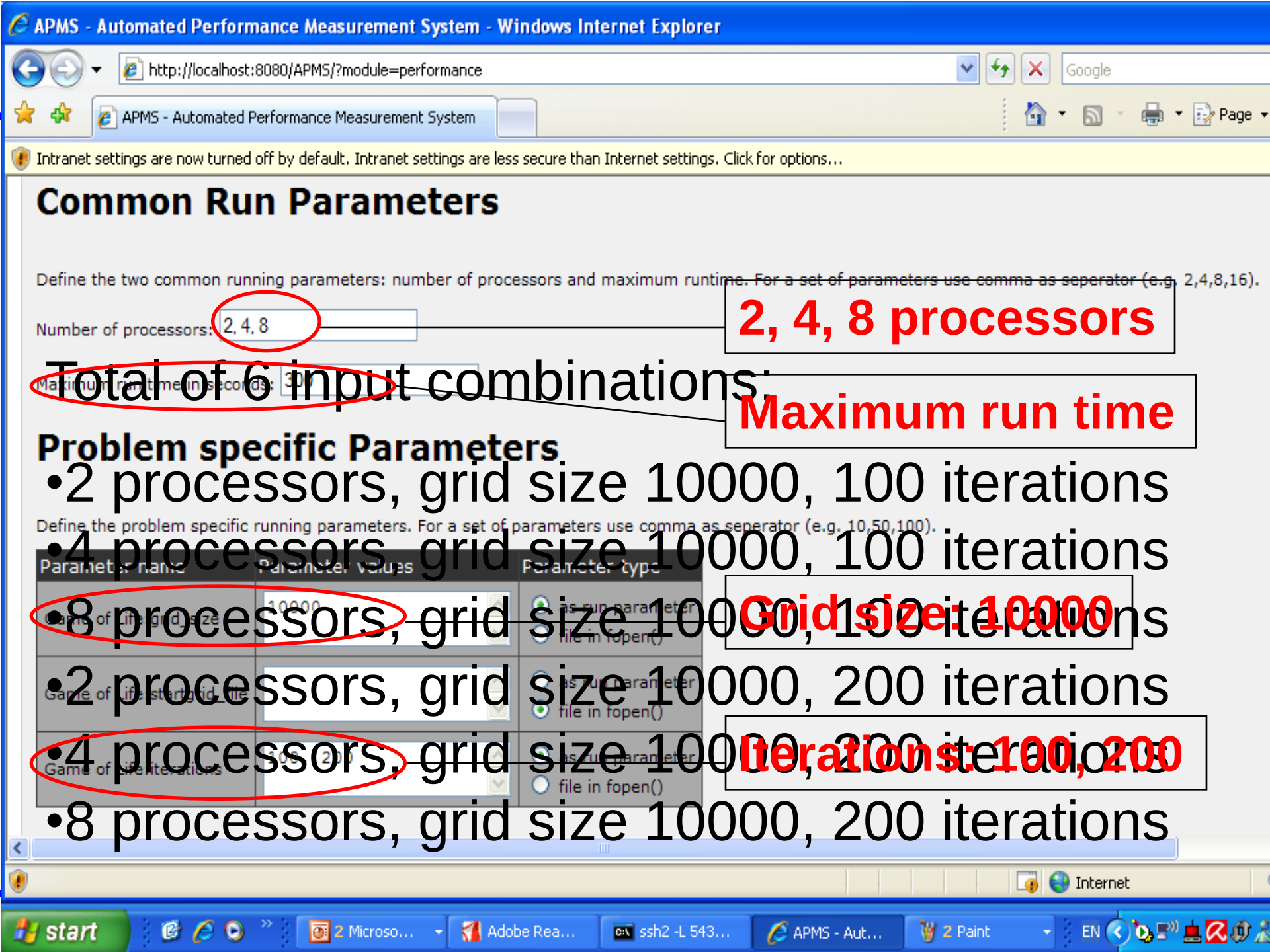


Example: Source Code Selection



Parameter Specification Language

- Specify values for Run Parameters:
  - Number of processors
  - Maximum runtime allowed
  - Problem Specific Parameters
- Supports multiple input values per parameter
  - Defined as a comma separated list
  - APMS generates multiple input sets
    - Using all possible combinations with multiple parameters
    - Runs the program on each input set



Total of 6 input combinations:

- 2 processors, grid size 10000, 100 iterations
- 4 processors, grid size 10000, 100 iterations
- 8 processors, grid size 10000, 100 iterations
- 2 processors, grid size 10000, 200 iterations
- 4 processors, grid size 10000, 200 iterations
- 8 processors, grid size 10000, 200 iterations

2, 4, 8 processors

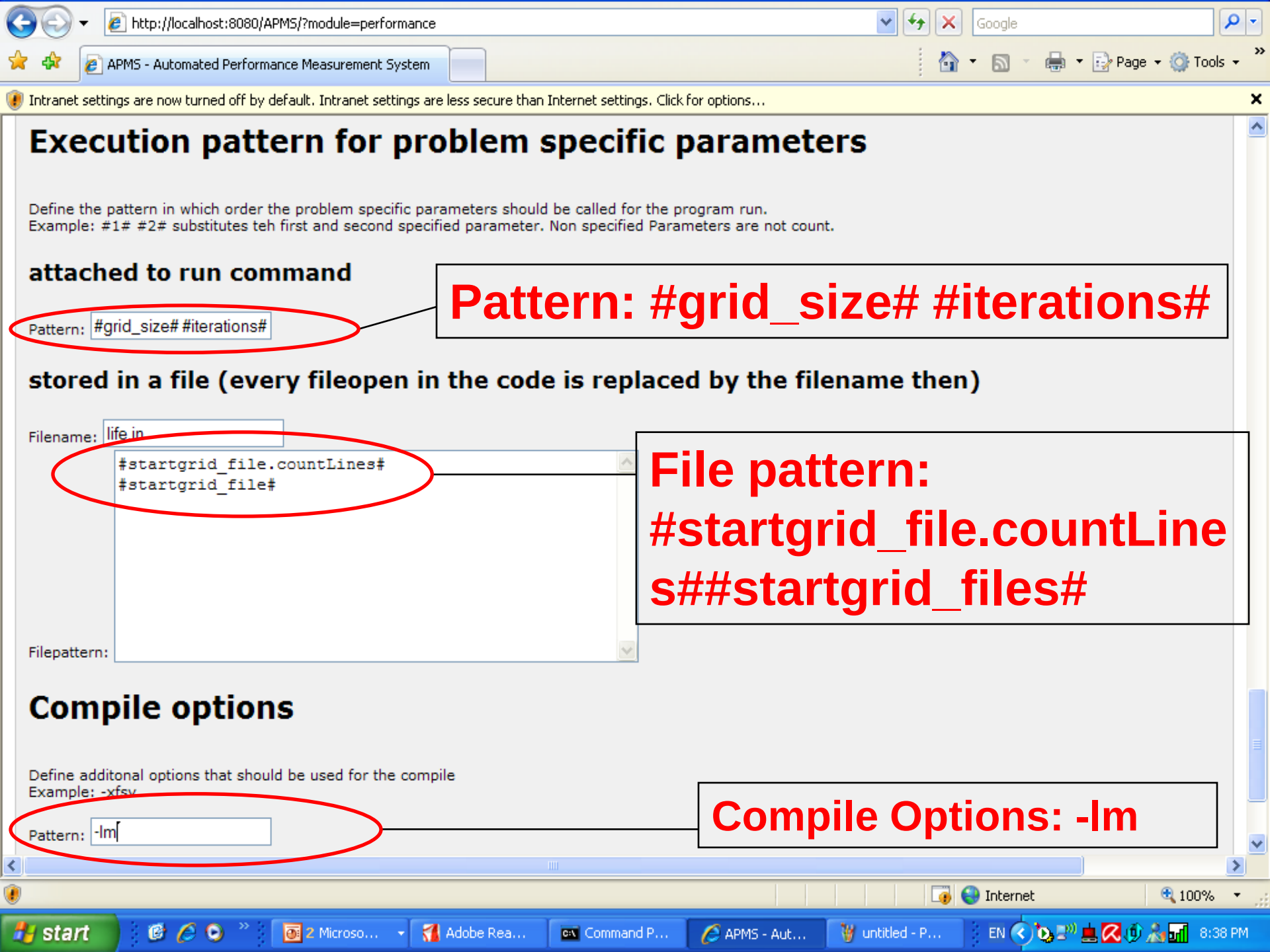
Maximum run time

Grid size: 10000

Iterations: 100, 200

Parameter Specification Language (Step 2)

- Specify order of parameters
  - Compile options pattern
  - Pattern attached to run command
  - Pattern to be inserted into input file


http://localhost:8080/APMS/?module=performance
APMS - Automated Performance Measurement System
Intranet settings are now turned off by default. Intranet settings are less secure than Internet settings. Click for options...
Execution pattern for problem specific parameters
Define the pattern in which order the problem specific parameters should be called for the program run.
Example: #1# #2# substitutes teh first and second specified parameter. Non specified Parameters are not count.
attached to run command
Pattern: #grid_size# #iterations#
Pattern: #grid_size# #iterations#
stored in a file (every fileopen in the code is replaced by the filename then)
Filename: life.in
#startgrid_file.countLines#
#startgrid_file#
Filepattern:
File pattern: #startgrid_file.countLines##startgrid_files#
Compile options
Define additonal options that should be used for the compile
Example: -xfsv
Pattern: -lm
Compile Options: -lm
Internet
100%
start
8:38 PM

Preliminary Studies

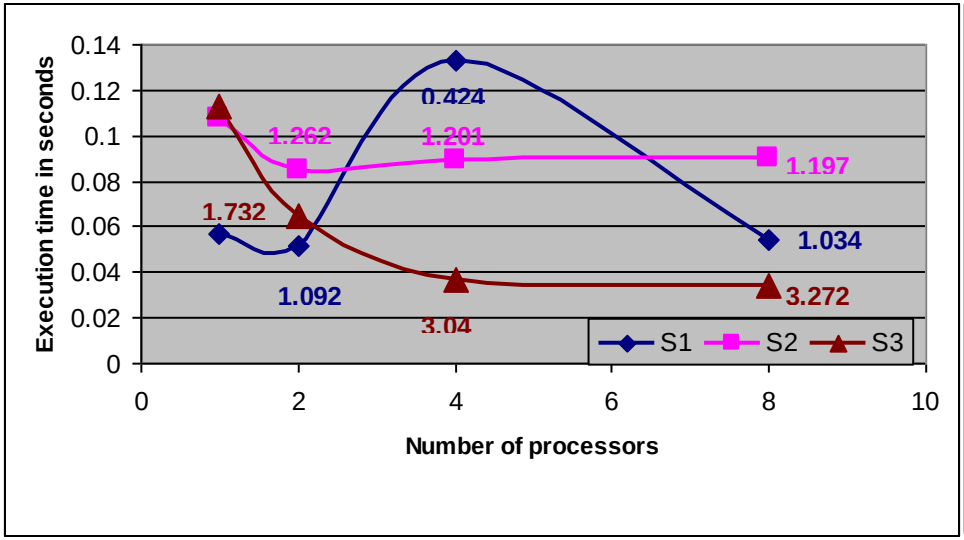
Do novices achieve speedup?

- Conjugate Gradient problem
- Class of 10 students
- 5 students used MPI
- 2 input sizes:
  - 8,000 x 8,000 matrix
  - 8,000,000 x 8,000,000 matrix

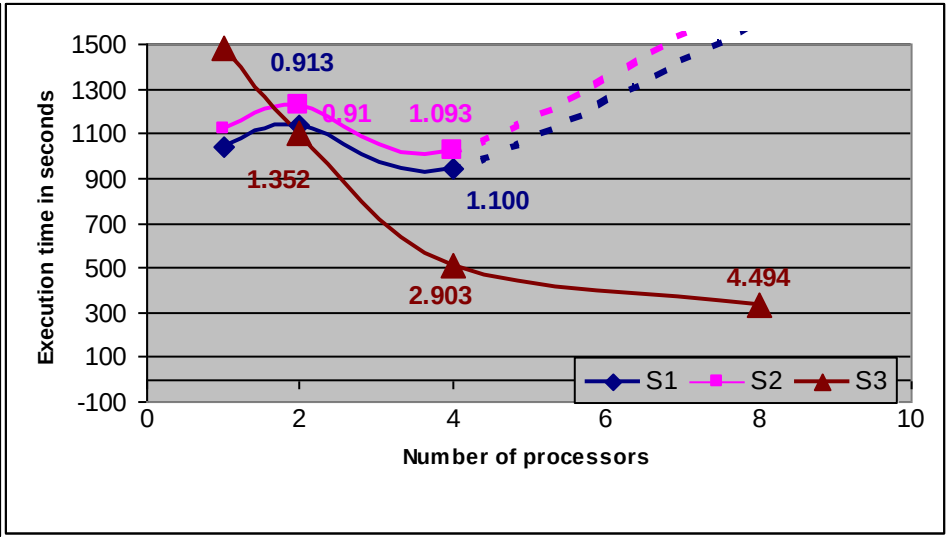
Do novices achieve speedup?

Problem: conjugate Gradient

Input matrix size: 8,000 x 8,000



Input matrix size: 8,000,000 x 8,000,000

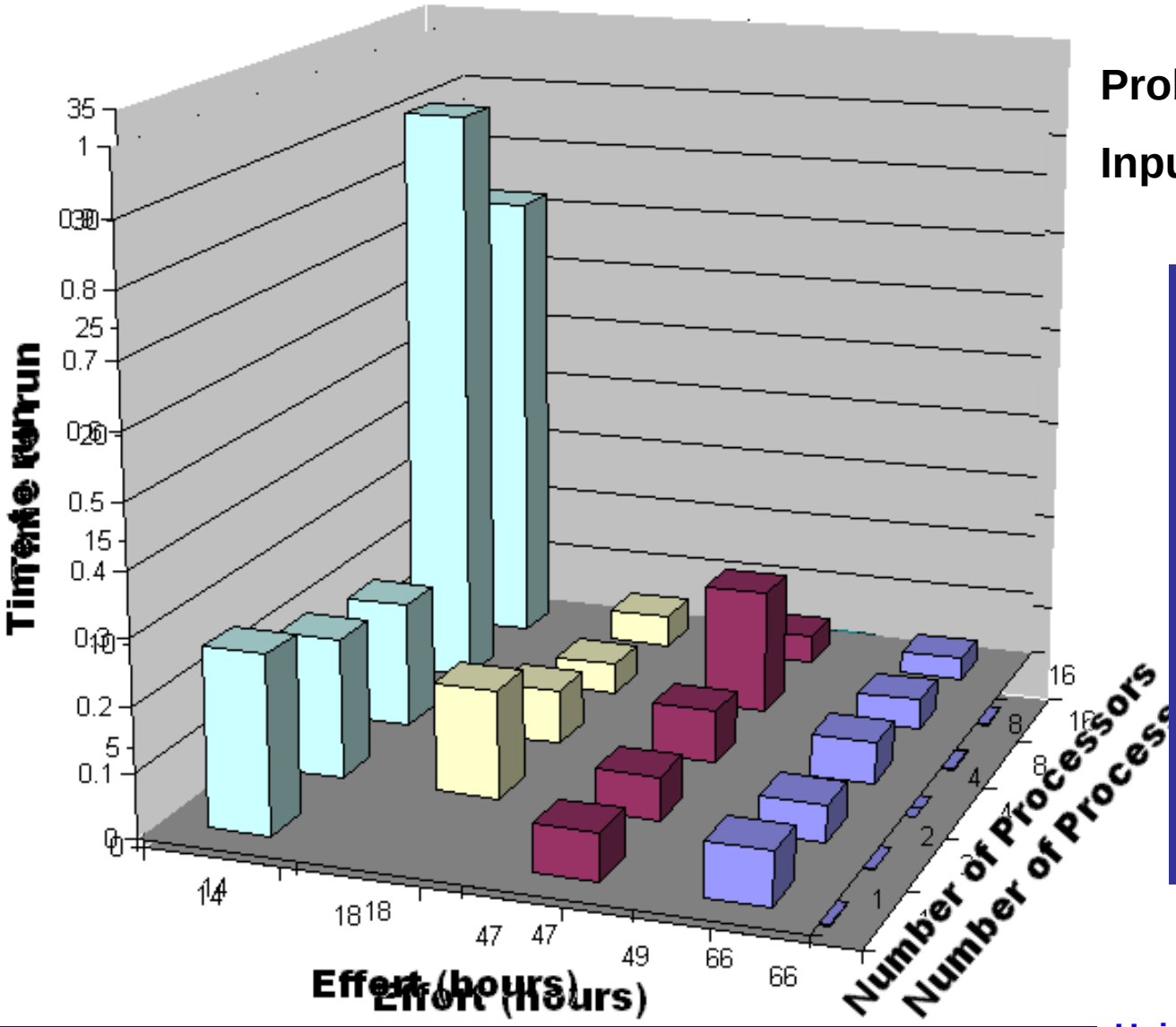


- **Some students achieve better speedup than others.**
- **Novice code that exhibits O.K performance deteriorates under the pressure of large input size, while novice code with good performance conserves its quality with large input size**

Does effort correlate with performance?

- Game of life problem
- Class of 10 students
- All students used MPI
- 1 input size:
  - 250 x 250 grid

Does effort correlate with performance?



Problem: Game of life

Input matrix size: 250 x 250

- **Sometimes performance is bad even with a lot of effort**
- **Two people with similar efforts can have widely different performance patterns**

Conclusions

- APMS
  - Makes it easy and fast to collect the needed performance data
  - Is flexible to allow measurement on any code
- Novices achieve speedup
  - Consistently good performance
  - Consistently bad performance
  - Oscillating performance based on input size and number of processors
- Determining the relationship between effort and performance is not easy
- Model refinement:
  - Regularity of work
  - Distribution of effort across activities
  - Background information

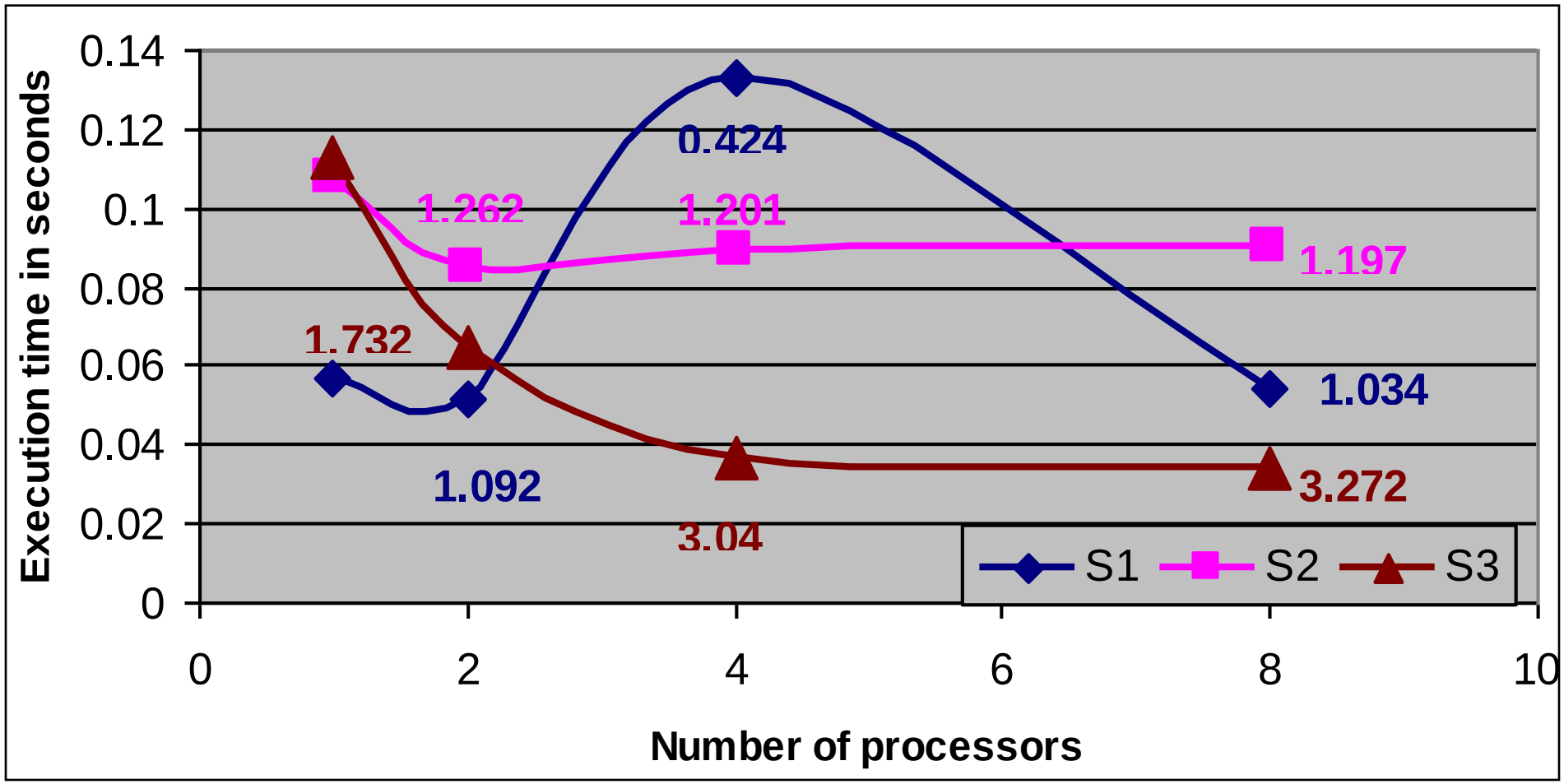
Thank you

- **Questions?**
- **Comments?**

- ***UMD***: Vic Basili, Marv Zelkowitz, Jeff Hollingsworth, Taiga Nakamura, Sima Asgari, Forrest Shull, Nico Zazworka, Rola Alameh, Daniela Suares Cruces
- ***UNL***: Lorin Hochstein
- ***MSU***: Jeff Carver
- ***UH***: Philip Johnson
- ***Professors teaching classes***: Alan Edelman [MIT], John Gilbert [UCSB], Mary Hall, Aiichiro Nakano, Jackie Charme [USC], Allan Snavely [UCSD], Alan Sussman, Uzi Vishkin [UMD], Ed Luke [MSU], Henri Casanova [UH], Glenn Leucke [ISU]
- ***Partners***: MIT Lincoln Labs, MIT, UCSD, UCSB, UMD, USC, FC-MD

Do novices achieve speedup?

Problem: conjugate Gradient **Input matrix size: 8000 x 8000**



Do novices achieve speedup?

Problem: conjugate Gradient **Input matrix size: 8000000 x 8000000**