Large Efficient Table-Top Teraflop Computing

Victor. Basili, Thiago Craveiro, Daniela Cruzes,
Kate Despain, Bill Dorland, Lorin Hochstein*,
Nico Zazworka, and Marvin Zelkowitz

University of Maryland in College Park
and
(* University of Nebraska, Lincoln)

Scientific Computing

- Problem: How to increase computational power for solving complex scientific problems?
- Solutions:
  - Increase speed of processing unit
  - If not powerful enough, build networks of processors (Traditional approach in building supercomputers – thousands of communicating processors)
    - Expensive to build
    - Expensive to use - Uses lots of power for computing and cooling
  - Alternative – Add inexpensive processors to current desktop machines to increase computational power.
    - Intel – Multicore processors
    - Use graphics processing units as general purpose computers (GPGPU)

This is the solution to be discussed today

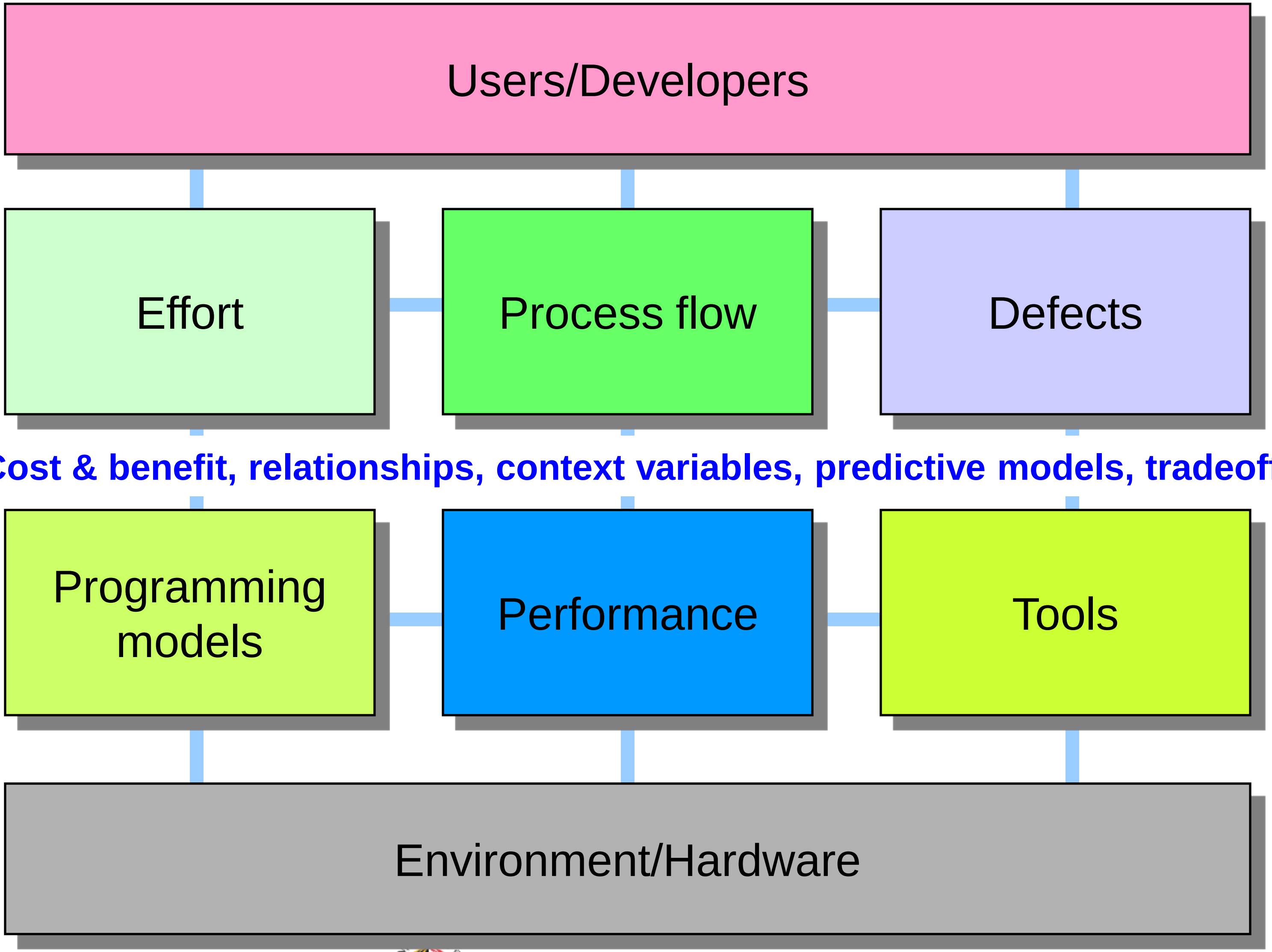
Productivity measures

- Related question: How effectively can we program these machines?
  - Traditionally the speed of the machine was measured in FLOPS (Floating Point Operations Per Second) on specific benchmark programs
    - Real programs rarely achieved those numbers
    - Often only 10-20% of peak performance
  - We have been studying programmer productivity in the High Performance Computing (HPC) domain as part of the DARPA High Productivity Computer System (HPCS) program from 2004-8 as a companion measure to machine performance
  - Can we apply those techniques to the problems of measuring productivity in the GPGPU domain.

Format for rest of talk

- Review aspects of our work on programmer productivity from the DARPA HPCS program
- Introduction to the GPGPU problem
- Initial work on this issue and some thoughts on how we intend to proceed

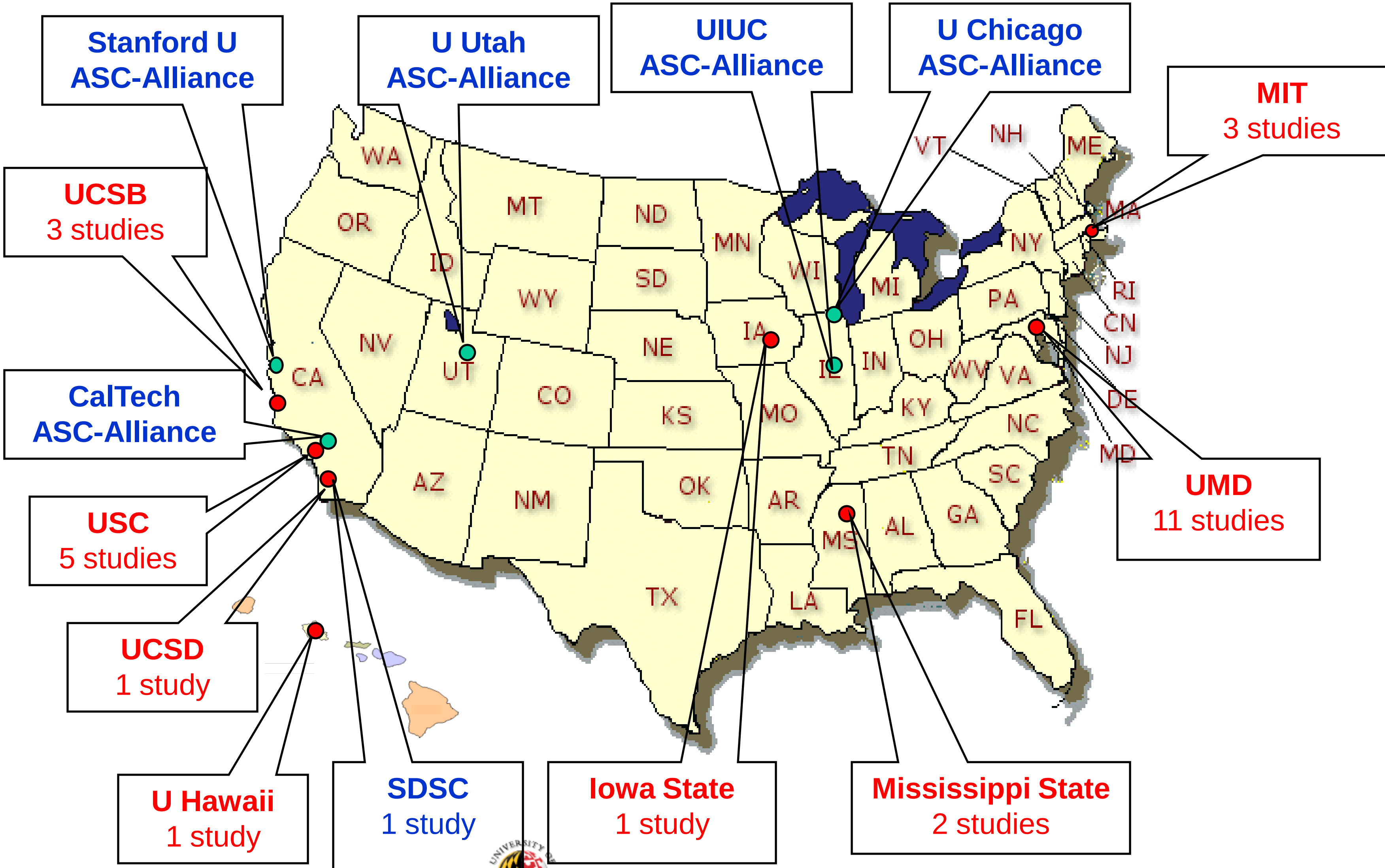
HPCS Areas of Study



Overall research process

- What: Performed several studies of programmers building HPC programs in various environments
  - Replicated studies with graduate students at various universities on a set of standardized programs
  - In-depth observational studies of a few individuals to understand their behavior in solving HPC problems
  - Interviews with developers on their experiences in building HPC codes
- How: Developed a series of tools for collecting development data
  - Effort data for programmers
  - Source files, edits, and test runs
  - System commands and execution times

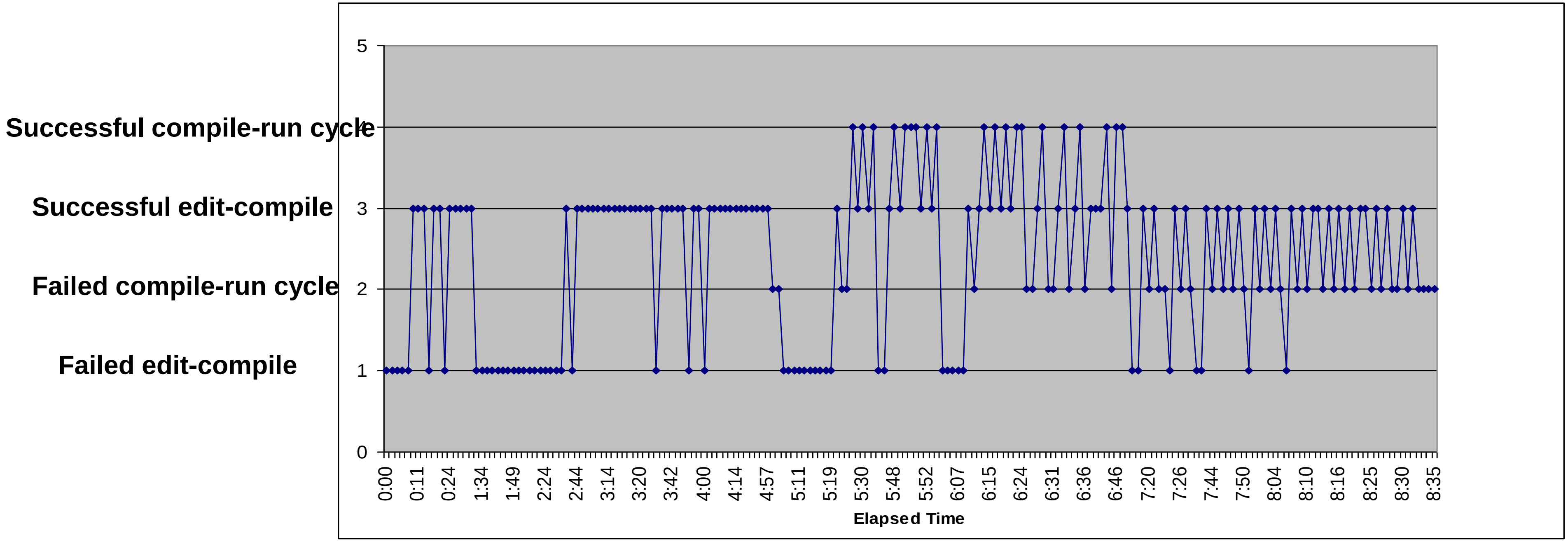
Studies conducted



Sample Results: Characterizing novices
(graduate students in classroom assignments)

- 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
- When performance is the goal:
  - Experts and students spend the same amount of time
  - Experts get significantly better performance
- No correlation between effort and performance

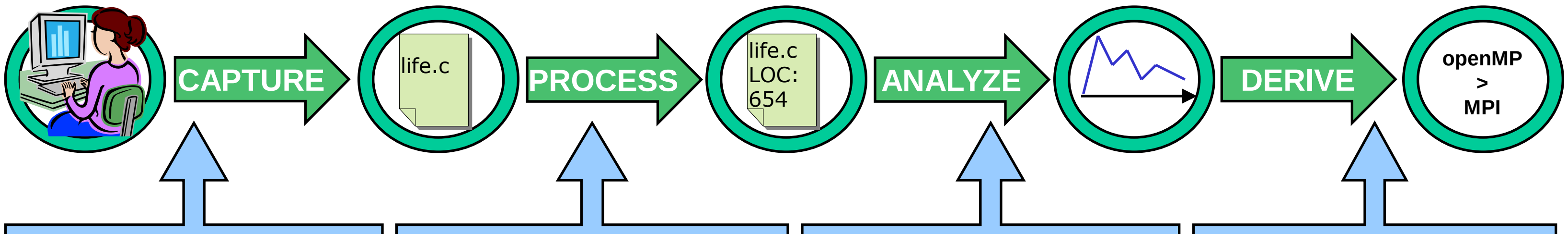
Results: Understanding workflow (Observational study)



Observation	A series of failed and successful Compile cycles with no runs	A series of failed and successful Compile-Run cycles	A series of successful Compile and failed Run cycles
Hypothesis	New code is being added and Compile Time defects being fixed	Run Time defects being fixed	Developer unable to fix defects
Truth (Interview)	Hypotheses were validated.		

Resulting Infrastructure Tools & Packages

For the hpcs studies we built a collection of tools



capture tools: help to gather data from study participants and join this data in our common data source - a relational DB

processing tools: calculate / post process data in the DB to retrieve non captured and higher level data

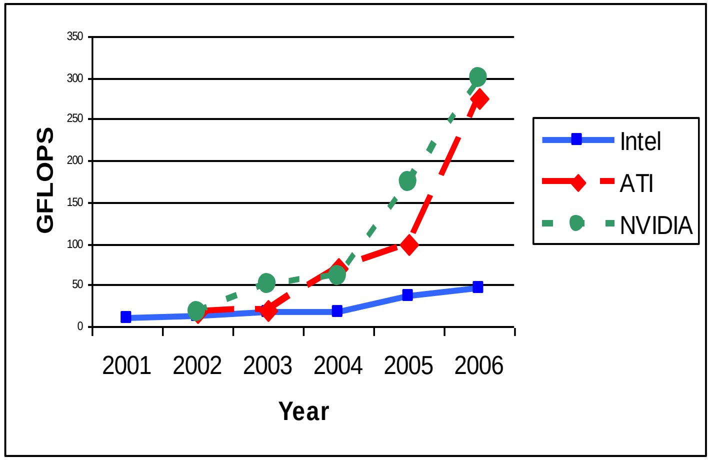
analyze tools: provide views on the DB in order to support the validation of hypotheses and to gain new insights

knowledge bases: present the derived knowledge of analyze processes

Information available at: http://hpcs.cs.umd.edu

GPGPU Solution

- High-end PCs use separate display processors (GPUs or graphics processing units) for manipulating data on the display for computational complex applications (e.g., video games)
- GPUs can be separately programmed for many tasks
- Speeds for GPUs are increasing faster than general CPU speeds



Question 1: Can GPUs be used to program solutions in the HPC domain?

- Can get today GPU boards with 512 or more GPUs

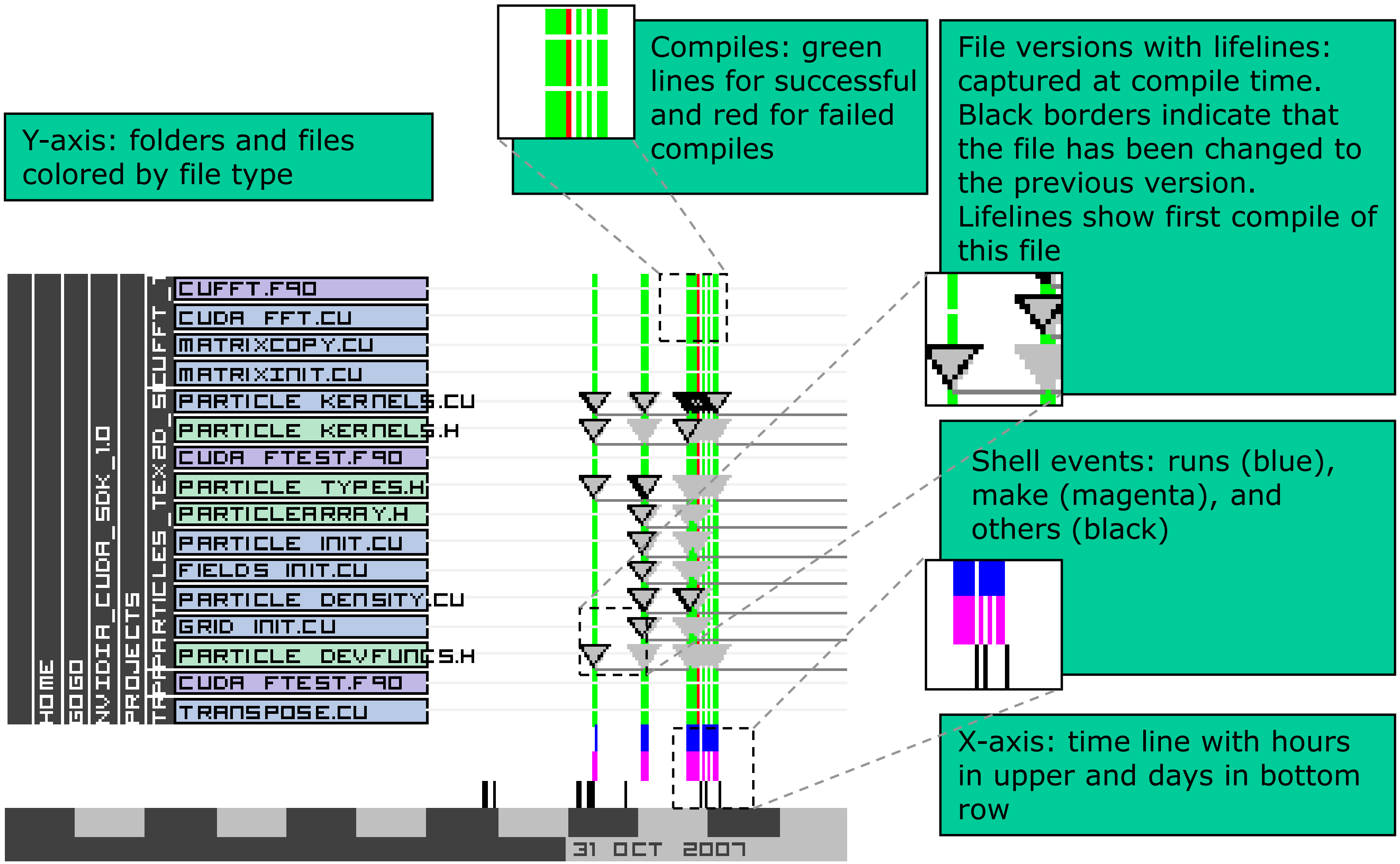
Question 2: Can we apply our approach in the HPCS domain to study GPGPU programming as well?

A group at the University of Maryland was porting an application from a multiprocessing system to a GPGPU system. This provided an environment for testing these ideas.

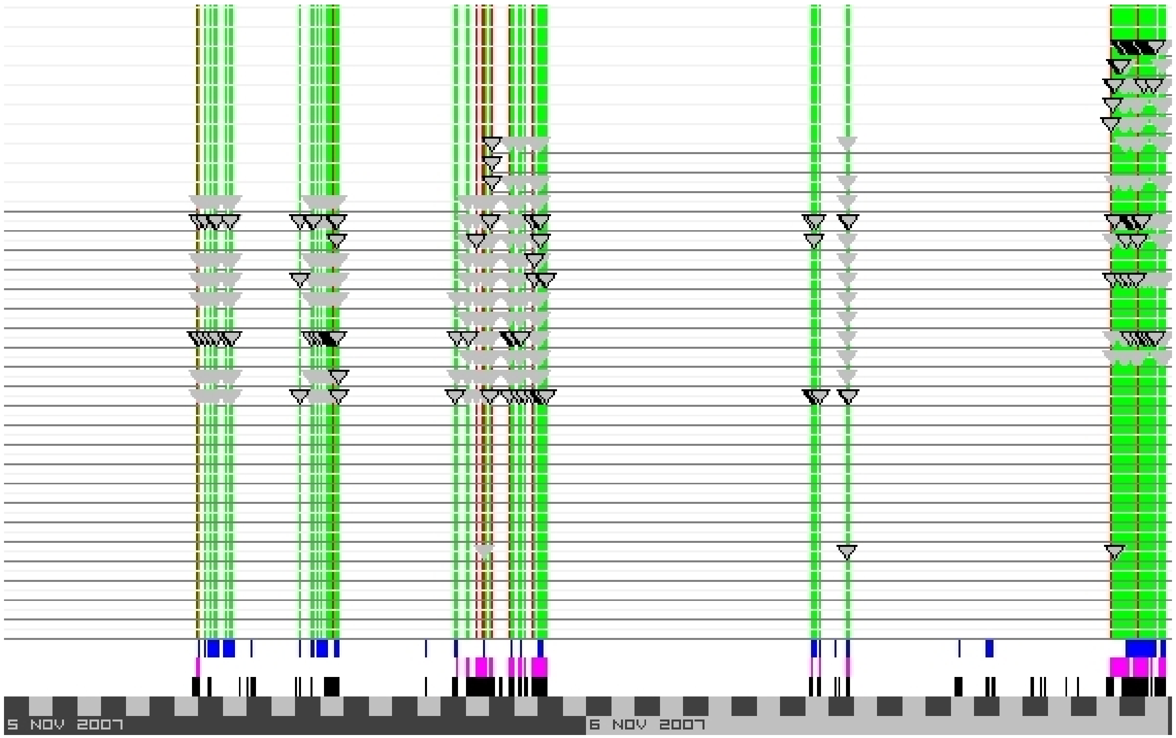
Initial issues under study

- Domain knowledge (how to solve the underlying problem in physics):
  - What distinguishes porting to a cluster from porting to a GPU?
  - What tools can aid scientists unfamiliar with GPUs when porting?
  - What tools help or are essential for software engineers using that methodology?
- Methodology understanding (how to study productivity issues):
  - What kind of methodology do you need to examine an on-going port?
  - How important are interviews for analysis?

CodeVizard – Software Evolution Visualization



Preliminary GPU study
(One week-port of rMHD code)



Observation	3 work sessions In first 2: No makes but runs In last: New files	High work density New files, focus on one Compiles Makes And runs
Hypothesis	First two phases: trying something new Third phase: getting first runs / earlier problems solved	Adding new component, dense and successful work points to error free development
Truth (Interview)	After meetings with colleagues he got the template code to run in the third phase. Adjustments were still necessary.	The subject ported his code to GPU in little time.

Scaling up: The weekly cycle steps

1. Process collected data – prior to interview
2. Pre-analysis of data – immediately before interview
3. Interview (semi-structured) developer
4. Post-analysis of data and interview

Question on Methodology

- Interviews in a longer study while it is in process instead of conducting them retrospectively?
  - Hypothesis: A week is a short enough time for the subject to remember details
  - Hypothesis: Regular code inspections (possible with tools) and interview techniques are effective necessary
- Experiences from each week can help improve both the methodology and the domain knowledge gain for the next one

Second GPU Case Study

- Characteristics:
  - Graduate student porting serial 2D MHD Fortran code to 3D on a GPU
  - Original used OpenMP. OpenMP removed from code and CUDA commands added
  - Used DevObject Fortran library; some work still had to be done in CUDA (kernels)
  - Parallelization of derivative and FFT calculation suspected to bring most speedup

Performance (derivatives)

- **Finding the derivative 1000 times for a 1024 by 1024 matrix using:**

Pointwise Matrix-Matrix multiplication takes: 0.9726562 secs

Pointwise Vector-Matrix multiplication takes: 0.8242188 secs

Scalar Constant cache + GPU integer math-Matrix mult. takes: 11.7148438 secs

Scalar in Shared memory + GPU integer math-Matrix mult. takes: 1.7734375 secs

- **Finding the derivative 1000 times for a 512 by 512 matrix using:**

Pointwise Matrix-Matrix multiplication takes: 0.2812500 secs

Pointwise Vector-Matrix multiplication takes: 0.2890625 secs

Scalar Constant cache + GPU integer math-Matrix mult. takes: 2.9765625 secs

Scalar in Shared memory + GPU integer math-Matrix mult. takes: 0.5117188 secs

- **Finding the derivative 1000 times for a 256 by 256 matrix using:**

Pointwise Matrix-Matrix multiplication takes: 0.1093750 secs

Pointwise Vector-Matrix multiplication takes: 0.1601562 secs

Scalar Constant cache + GPU integer math-Matrix mult. takes: 0.8085938 secs

Scalar in Shared memory + GPU integer math-Matrix mult. takes: 0.1914062 secs

Preliminary results: Domain knowledge

- Most defects are related to environment (CUDA / DevObject), some to memory (shared memory usage)
- Workflow:
  - A lot of prototyping and testing/benchmarking before creating final code
  - Parallelization of serial 2D version first, then addition and parallelization of 3D, one attempt using parallel “scan” primitive for total energy sum calculation, then final physics code
  - Reuse of code consisted of a big increment in one file + small increment in others
  - Most of the time spent in understanding and adapting environment (CUDA / DevObject / reused code)

Preliminary results: Methodology

- Defects: Hard to recognize patterns judging from syntax errors alone
- Interviews:
  - Structured interview questions about goal and priority changes (most occurring after meetings) turn out to be **very** important
  - Unstructured questions hard to formulate without clarification / screenshots, require a lot of preparation
  - Also they are not easy to answer in a few words, so the subject also needs a long time to explain
  - Interview too short to cover more than one aspect per week (defects, effort, workflow,…)

Conclusions

- Still at preliminary stage for understanding effectiveness of GPGPU programming
- Methodology understanding:
  - Need Improvement of tools (system view/code view annotation in CodeVizard)
  - Need larger-scale and classroom experiments on defects, effort & performance
  - Need refinement of interview templates for effort and defects and creation of new ones for other HPC research goals

Goal: Better understanding of the issues in programming GPUs as a substitute for HPC machines.