1. Introduction
As part of the Department of Energy’s Scientific Discovery through Advanced Computing program, Rice University is developing HPCTOOLKIT [1], a performance toolkit for accurately measuring and pinpointing performance bottlenecks on emerging petascale computer systems.

HPCTOOLKIT consists of components for measuring the performance of fully-optimized executables of parallel programs, analyzing application binaries to correlate measurements with program structure, and novel analysis techniques for pinpointing performance bottlenecks. We designed HPCTOOLKIT to:

- **Work at binary level for language independence.** This enables HPCToolkit to support measurement and analysis of multi-lingual codes with external binary-only libraries.

- **Profile rather than adding code instrumentation.** Sample-based profiling is less intrusive than code instrumentation and requires only very modest data volume.

- **Collect and correlate multiple performance metrics.** Performance problems typically cannot be diagnosed with only one species of event.

- **Compute derived metrics to aid analysis.** Synthetic metrics, such as memory bandwidth consumed, often provide insight for optimization.

- **Attribute costs very precisely.** HPCTOOLKIT is unique in its ability to associate measurements with dynamic calling context, loops, and inlined code.

We have developed several novel performance analysis techniques that provide deep insights into application performance on parallel platforms.

(i) **Pinpointing scalability bottlenecks in SPMD codes.** By using differential analysis of multiple parallel executions at different scales, HPCTOOLKIT can pinpoint scalability bottlenecks all the way down to individual source lines in the program and the calling context in which the bottlenecks arose. Surprisingly, one can do this for production runs of fully-optimized applications with run-time measurement overhead of only a few percent for arbitrary SPMD parallel programs, including applications written using MPI, UPC, and Co-array Fortran [2].

(ii) **Identifying performance bottlenecks in multithreaded codes.** In 2008, we extended HPCTOOLKIT with two novel techniques for pinpointing, quantifying, and explaining performance bottlenecks in multithreaded programs. Our approach supports diagnosis of two types of inefficiencies: insufficient parallelism (which might be due to load imbalance or serialization) and parallelization overhead.

To pinpoint and quantify insufficient parallelism in executions of multithreaded programs, we measure two quantities at sample points: the number of threads performing useful work \( W \) and those that are idle \( I \). If a sample event occurs in a thread that is idle, we ignore it. When a sample event occurs in a thread that is actively working, the thread records...
one sample in a metric representing the thread’s work and in a second metric, the thread records a fractional sample $I/W$ to charge it a proportional share of its responsibility for not keeping the idle processors busy at that moment at that point in the program. From this, we can compute a metric representing the loss of parallelism and inefficiency associated with each program context. We use post-mortem binary analysis to attribute samples to parallelization overhead using information recorded during compilation.

(iii) Showing temporal evolution of parallel codes. Space-time diagrams based on traces of communication events are commonly used to understand the temporal evolution of a parallel execution. In 2008, we developed hpctraceview, which presents a novel space-time view of parallel program executions based on traces of asynchronous call stack samples. Our viewer can display the space time evolution at multiple levels of abstraction: one can select the call stack depth for the presentation using a slider. Figure 1 shows a snapshot of hpctraceview displaying the temporal evolution of an MPI program for turbulent combustion program on 8 processors. The center panel shows both context and detailed views of how the execution evolves as time advances left to right. The rightmost pane shows the call path for a selected sample. By reducing the timeline to a single pixel high, hpctraceview can readily scale to show views of hundreds of processors.

2. Current status
We are currently using HPCToolkit to analyze application performance on Opteron-based Linux clusters. This summer, it will be installed on TACC’s 500TF Ranger system. Prototype versions are operational on both Cray XT and Blue Gene systems and will be released as soon as kernel bugs on these systems are resolved.

References