Performance Analysis Tools for Partitioned Global-Address-Space Programming Models

Adam Leko[1], Hung-Hsun Su[1], Dan Bonachea[2], Max Billingsley III[1],
Hans Sherburne[1], Bryan Golden[1], and Alan D. George[1]

[1]Department of Electrical and Computer Engineering
University of Florida
{leko, su, billingsley, sherburne, golden, george}@hcs.ufl.edu

[2]Department of Computer Science
University of California at Berkeley
bonachea@cs.berkeley.edu

Extended Abstract (Oral Presentation/Demonstration)

The Partitioned Global-Address-Space (PGAS) programming model provides important productivity advantages over traditional parallel programming models. However, due to their implementation complexity, languages and libraries using PGAS models currently have little to no support from existing performance tools. We have designed the Global Address Space Performance tool interface (GASP) that is flexible enough to be used with any PGAS model, while simultaneously allowing existing performance tools to leverage their existing tool's infrastructure to quickly add support for programming languages and libraries using PGAS models. Additionally, we have developed Parallel Performance Wizard (PPW), a performance tool focused towards PGAS models, which provides a strong proof-of-concept for the GASP interface.

GASP

The GASP interface was born out of the need to support several different PGAS models in our PPW tool. As we studied different PGAS implementations we soon realized that large development complexity would be necessary to support different PGAS model implementations, as the flexibility of PGAS models allow for many implementation techniques. This development complexity is prohibitive to performance tool developers, inhibiting wide-scale support for PGAS languages in performance tools.

To encourage performance tool support for PGAS models, we created a simple, model-independent, portable and flexible performance tool interface based on callbacks that is especially useful for capturing performance from programs using PGAS models. In our talk, we will give a high-level overview of the GASP tool interface, showing how the interface has been successfully implemented in Berkeley UPC and how the interface can be easily applied to upcoming high-productivity programming languages such as X10, Fortress, and Chapel. The presentation will include empirical results demonstrating the instrumentation overheads of GASP for several UPC applications, and the scalability of the approach. A major goal of this presentation is to raise awareness for the GASP interface so that it gains momentum for both performance tool developers and PGAS model implementers.

PPW

PPW is a performance tool specifically geared towards maximizing user productivity for tuning programs using PGAS models. The tool features easy-to-use compiler wrapper scripts that control the instrumentation process, and support for profile and trace data. Additionally, PPW provides a cross-platform user interface with PGAS-specific visualizations, including an array layout visualization feature for UPC programs, a communication visualization that gives a detailed breakdown of inter-thread communication (Figure 1), and profile data viewers that give a high-level overview of program performance (Figure 2). The tool also includes a SLOG-2 trace export for use with the scalable Jumpshot trace viewer (Figure 3). Finally, PPW integrates support for hardware counters through the PAPI library so users may perform detailed analysis of architectural behavior in computational regions of their applications.

As part of our talk, we will give a brief demonstration of the PPW tool, showing how the tool uses the GASP interface and how the visualizations aid users in pinpointing bottlenecks in their applications.

Project References

PPW: http://www.hcs.ufl.edu/ ppw - includes alpha release of PPW tool
GASP: http://www.hcs.ufl.edu/upc/gasp - includes draft GASP specification
Berkeley UPC: http://upc.lbl.gov/

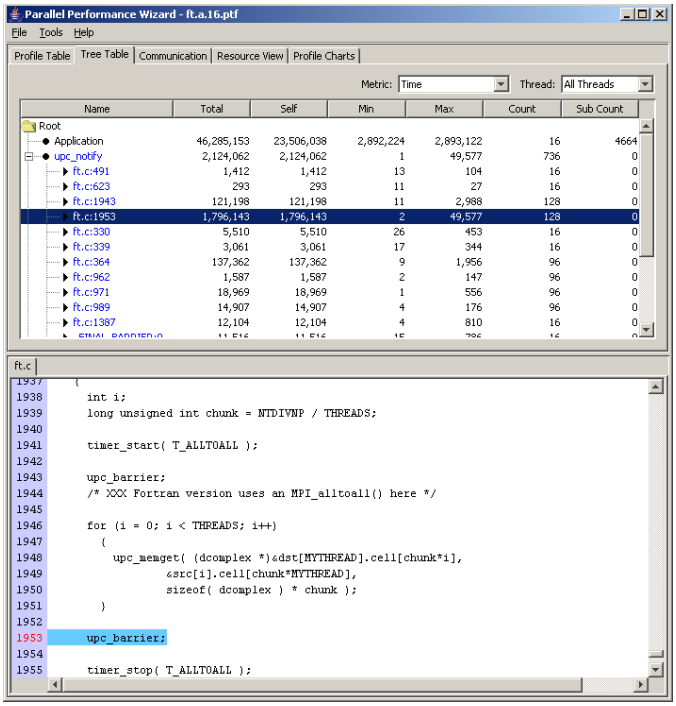

Figure 1: Calltree profile viewer. The display shows profile data for a 16-node run of the NAS FT benchmark alongside the original source code.

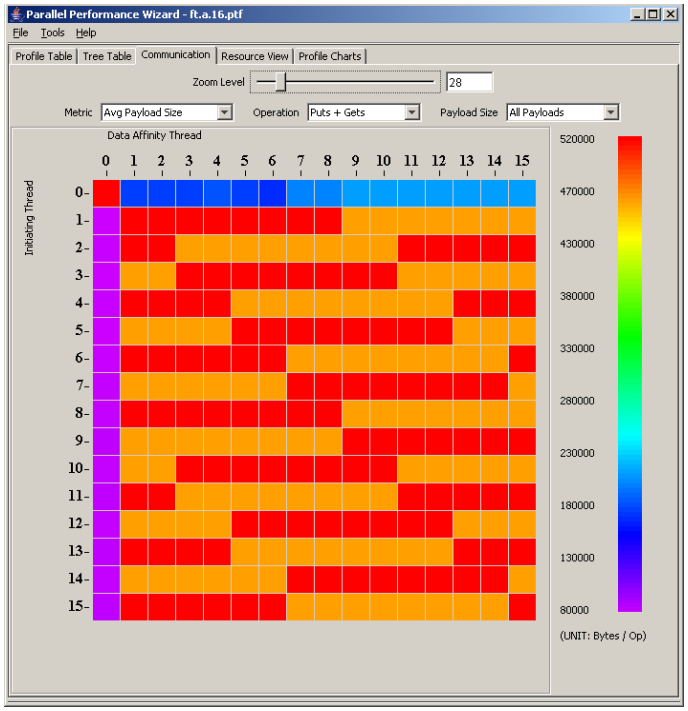

Figure 2: Communication visualization. This viewer allows one to view payload size statistics for gets and puts broken down by payload size.

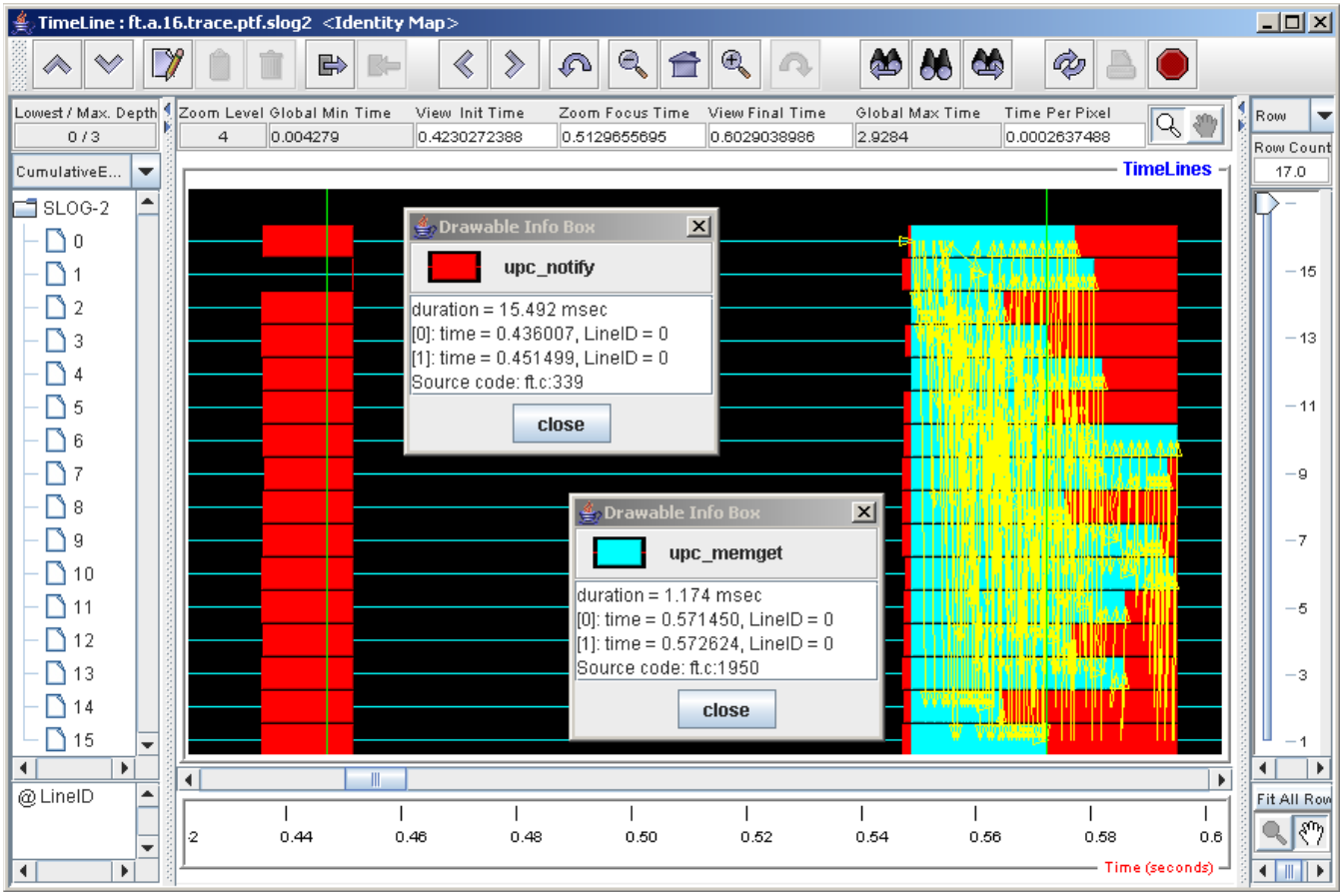

Figure 3: SLOG-2 export viewed in the Jumpshot viewer. Right-clicking on events in the timeline brings up detailed information about that event, including source code information. This type of viewer is very useful for visually detecting load imbalances, such as the imbalance of time spent in `upc_notify` in thread 1 in this example.