Status of Krell Tools Built using Dyninst/MRNet

Paradyn Week 2013
Madison, Wisconsin
April 30, 2013
Presenters

- Jim Galarowicz, Krell
- Don Maghrak, Krell

Larger team

- William Hachfeld, Dave Whitney, Dane Gardner: Krell
- Martin Schulz, Matt Legendre, Chris Chambreau: LLNL
- Jennifer Green, David Montoya, Mike Mason, Phil Romero: LANL
- Mahesh Rajan, Anthony Agelastos: SNLs
- Dyninst group:
  - Bart Miller, UW and team
  - Jeff Hollingsworth, UMD and team
- Phil Roth, Michael Brim: ORNL
Outline

- Welcome
  1. Open SpeedShop overview and status
  2. Component Based Tool Framework overview and status
  3. SWAT (Scalable Targeted Debugger for Scientific and Commercial Computing) DOE STTR Project Status
  4. GPU Support DOE SBIR Project Status
  5. Cache Memory Analysis DOE STTR Project Status
  6. Parallel GUI Tool Framework DOE SBIR Project Status
  
- Questions
Open|SpeedShop™

COMPONENT BASED TOOL FRAMEWORK: CBTF

Open|SpeedShop

(www.openspeedshop.org)

Paradyn Week 2013

April 20, 2013
What is Open|SpeedShop?
- HPC Linux, platform independent application performance tool
- Linux clusters, Cray, Blue Gene platforms supported

What can Open|SpeedShop do for the user?
- `pcsamp`: Give lightweight overview of where program spends time
- `usertime`: Find hot call paths in user program and libraries
- `hwc,hwctime,hwcsamp`: Give access to hardware counter event information
- `io,iot`: Record calls to POSIX I/O functions, give timing, call paths, and optional info like: bytes read, file names...
- `mpi,mpit`: Record calls to MPI functions. give timing, call paths, and optional info like: source, destination ranks, ..... 
- `fpe`: Help pinpoint numerical problem areas by tracking FPE
Maps the performance information back to the source and displays source annotated with the performance information.

> openss -cli -f smg2000-pcsamp.openss
openss>>Welcome to OpenSpeedShop 2.0.2
openss>>expview

<table>
<thead>
<tr>
<th>Function (defining location)</th>
<th>% of CPU Time</th>
<th>Time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>hypre_SMGResidual</td>
<td>43.060498221</td>
<td>3.630000000</td>
</tr>
<tr>
<td>hypre_CyclicReduction</td>
<td>33.926453144</td>
<td>2.860000000</td>
</tr>
<tr>
<td>hypre_SemiRestrict</td>
<td>33.926453144</td>
<td>2.860000000</td>
</tr>
<tr>
<td>hypre_SemiInterp</td>
<td>32.1470937</td>
<td>2.491103203</td>
</tr>
<tr>
<td>opal_progress</td>
<td>1.779359431</td>
<td>1.779359431</td>
</tr>
</tbody>
</table>

How you run your application outside of OSS”

openss -f smg2000-pcsamp.openss for GUI

openss –cli –f smg2000-pcsamp.openss for CLI (command line)
Update on status of Open|SpeedShop

- Continued to focus more on CBTF the past year
- Completed port to Blue Gene Q
  - Static executables using osslink
  - Dynamic (shared) executable using osspcsamp, ossusertime, etc.
- Added functionality to Open|SpeedShop
  - Added MPI File I/O support to MPI experiment.
  - Keeping up with components like: libunwind, papi, dyninst, libmonitor...
  - Derived metric support: arithmetic on gathered performance metrics
  - More platforms, users & application exposure -> more robust
- New CBTF component instrumentor for data collection
  - Leverages lightweight MRNet for scalable data gathering and filtering.
  - Uses CBTF collectors and runtimes
  - Passes data up the transport mechanism, based on MRNet
  - Provides basic filtering capabilities currently
Future Experiments by End of 2013

- **New Open|SpeedShop experiments under construction**
  - **Lightweight I/O experiment (iop)**
    - Profile I/O functions by recording individual call paths
      - Rather than every individual event with the event call path, (io and iot).
      - More opportunity for aggregation and smaller database files
    - Map performance information back to the application source code.
  - **Memory analysis experiment (mem)**
    - Record and track memory consumption information
      - How much memory was used – high water mark
      - Map performance information back to the application source code
  - **Threading analysis experiment (thread)**
    - Report statistics about pthread wait times
    - Report OpenMP (OMP) blocking times
    - Attribute gathered performance information to proper threads
    - Thread identification improvements
      - Use a simple integer alias for POSIX thread identifier
    - Report synchronization overhead mapped to proper thread
    - Map performance information back to the application source code
Scaling Open|SpeedShop

- Open|SpeedShop designed for traditional clusters
  - Tested and works well up to 1,000-10,000 cores
  - Scalability concerns on machines with 100,000+ cores
  - Target: ASC capability machines like LLNL’s Sequoia (20 Pflop/s BG/Q)

- Component Based Tool Framework (CBTF)
  - [http://ft.ornl.gov/doku/cbtfw/start](http://ft.ornl.gov/doku/cbtfw/start)
  - Based on tree based communication infrastructure
  - Porting O|SS on top of CBTF

- Improvements:
  - Direct streaming of performance data to tool without writing temporary raw data I/O files
  - Data will be filtered (reduced or combined) on the fly
  - Emphasis on scalable analysis techniques

- Initial prototype exists, working version: Mid-2013
  - Little changes for users of Open|SpeedShop
  - CBTF can be used to quickly create new tools
  - Additional option: use of CBTF in applications to collect data
What UW/UMD software is used in Open|SpeedShop?

- **symtabAPI**
  - For symbol resolution on all platforms

- **instructionAPI, parseAPI**
  - For loop recognition and details
    - This work is in progress

- **dyninstAPI**
  - For dynamic instrumentation and binary rewriting
    - Includes the subcomponents that comprise “Dyninst”.
    - Inserts performance info gathering collectors and runtimes into the application.

- **MRNet**
  - Transfer data from application level to the tool client level.
  - Filtering of performance data on the way up the tree.

Keeping up with the releases and pre-release testing

- At release level 8.1.1
Component Based Tool Framework (CBTF)

Paradyn Week 2013

April 20, 2013
What is CBTF?
- A Framework for writing Tools that are Based on Components.
- Consists of:
  - Libraries that support the creation of reusable components, component networks (single node and distributed) and support connection of the networks.
  - Tool building libraries (decomposed from O|SS)

Benefits of CBTF
- Components are reusable and easily added to new tools.
- With a large component repository new tools can be written quickly with little code.
- Create scalable tools by virtue of the distributed network based on MRNet.
- Components can be shared with other projects
CBTF uses a transport mechanism to handle all of its communications.

CBTF uses MRNet as its transport mechanism
- Multicast/Reduction Network
- Scalable tree structure
- Hierarchical on-line data aggregation

CBTF views MRNet as “just” another component.
CBTF Networks

- Three Networks where components can be connected
  - Frontend, Backend, multiple Filter levels
  - Every level is homogeneous

- Each Network also has some number of inputs and outputs.

- Any component network can be run on any level, but logically
  - Frontend component network
    - Interact with or Display info to the user
  - Filter component network
    - Filter or Aggregate info from below
    - Make decisions about what is sent up or down the tree
  - Backend component network
    - Real work of the tool (extracting information)
What can this framework be used for?

CBTF is flexible and general enough
- To be used for any tool that needs to “do something” on a large number of nodes and filter or collect the results.

Sysadmin Tools
- Poll information on a large number of nodes
- Run commands or manipulate files on the backends
- Make decisions at the filter level to reduce output or interaction

Performance Analysis Tools
- Massively parallel applications need scalable tools
- Have components running along side the application

Debugging Tools
- Use cluster analysis to reduce thousands (or more) processes into a small number of groups
CBTF Related and Next Steps

❖ Tool startup investigations (Libi, launchmon)

❖ Continuing porting to Cray and Blue Gene
  ➢ Cray
    • Working, but needs some further automation for node allocation
  ➢ Blue Gene
    • Delayed, because lightweight MRNet does not currently work on BG/Q
    • Investigation with Matt Legendre, LLNL, on an alternative way to transfer performance information from the application to the CBTF/OSS tool.

❖ Add more advanced data reduction filters
  ➢ Cluster analysis
  ➢ Data matching techniques: keep a representative rank/thread

❖ Full Open|SpeedShop integration

❖ Completed Phase I DOE SBIR to research and add performance analysis support for GPU/Accelerators
Scalable Targeted Debugger for Scientific and Commercial Computing (SWAT) STTR Project

Paradyn Week 2013

April 20, 2013
What is SWAT?
- A Commercialized version of the STAT debugger primarily developed by LLNL/UW
- Attach to a hung job, find all call paths and expose the outliers.

UW and Argo Navis* teamed together on STTR to:
- Port SWAT to more platforms
- Test and extend the stack walking component used by SWAT, the StackwalkerAPI to work with more compilers, platforms, ...
  - This was done
- Enhance the GUI so that it is portable, robust, and easy to use.
  - New GUI was written based on the Parallel Tools GUI Framework (PTGF)
- Develop more advanced call tree reduction algorithms
- Improve SWAT’s ability to display complex stack trees

Uses StackWalkerAPI and MRNet

Looking for new funding and marketing opportunities for SWAT.

*Commercial entity associated with Krell
Open | SpeedShop™

COMPONENT BASED TOOL FRAMEWORK: CBTF

Open|SpeedShop Support GPU SBIR Phase I Project

Paradyn Week 2013

April 20, 2013
GPU support: CBTF & OpenSpeedShop

- Argo Navis* GPU DOE SBIR phase I
  - Prototype application profiling support for GPUs into OpenSpeedShop

- Using the CUDA and PAPI Cupti interfaces

- These were the goals we proposed for the GPU SBIR:
  - Report the time spent in the GPU device (when exited - when entered).
    - Completed
  - Report the cost and size of data transferred to and from the GPU.
    - Completed
  - Report information to help the user understand the balance of CPU versus GPU utilization.
    - Close to completion
  - Report information to help the user understand the balance between
    - The transfer of data between the host and device memory and the execution of computational kernels.
      - Have info to derive this, need to create the views.
  - Report information to help the user understand the performance of the internal computational kernel code running on the GPU device.
    - Close to completion

*Commercial entity associated with Krell
GPU support: CBTF & OpenSpeedShop

- Because transitioning Open|SpeedShop to use CBTF to collect performance data.
  - GPU collection capabilities were added to the CBTF collector set. Makes the functionality available in CBTF as well.

- Rudimentary views are available.
  - Info external to GPU displays based on I/O tracing collector view
  - Info internal to GPU displays based on the hwc sampling collector view

- Current status:
  - Collection of external GPU kernel statistics is completed
  - Working on gathering information about the GPU kernels themselves.
  - Looking for new funding opportunities for further GPU related development, as we did not win phase II funding.
    - CLI and GUI view work needed.

*Commercial entity associated with Krell*
Cache Memory Analysis STTR Phase I Project (active)

Paradyn Week 2013

April 20, 2013
Automated Cache Performance Analysis

Automated Cache Performance Analysis and Optimization in Open|SpeedShop

- Teamed with Kathryn Mohror and Barry Roundtree, LLNL
- Use Precise Event-Based Sampling (PEBS) counters
- With the newest iteration of PEBS technology
  - Cache events can be tied to a tuple of:
    - Instruction pointer
    - Target address (for both loads and stores)
    - Memory hierarchy and observed latency
- With this information we can analyze Cache usage for:
  - Efficiency of regions of code
  - How these regions interact with particular data structures
  - How these interactions evolve over time.

Short term, research focus:
- Performance analysis: understanding and optimizing the behavior of application codes related to their memory hierarchy.

Long term, research focus: Automation
Parallel Tools GUI Framework (PTGF)  
Phase I Project (active)

Paradyn Week 2013

April 20, 2013
Parallel Tools GUI Framework Goals:

- Facilitate the rapid development of cross-platform user interfaces for new and existing parallel tools.
- Target a stable version of Qt4 which is currently available on many existing clusters. It is forward compatible with Qt5.
- Provide abstracted visualizations for easy inclusion in multiple parallel tools. These abstracted visualizations will accept a simple dataset.
  - The visualization plugins will also act as dynamic libraries, which can be easily extended by tool developers looking to specialize a particular view.
- Provide a scalable design/model which will allow tools with very large datasets to be used effectively within the PTGF.
- Provide a standardized interface such that users will find enough similarities between tools to make learning additional ones easier.
- Provide facilities for user learning of a new parallel tool from within PTGF, and the ability to link to online resources.
Questions

- Jim Galarowicz
  - jeg@krellinst.org

- Don Maghrak
  - dpm@krellinst.org

- Questions about Open|SpeedShop or CBTF
  - oss-questions@openspeedshop.org