Cray HPCS Productivity Features

Joint NERSC/OLCF/NICS
Cray XT5 Workshop
February 1-3, 2010

Margaret Cahir
Cray, Inc.
Agenda

- Background on the Productivity Efforts
- 2 Productivity Tools/Features
  - ATP (Abnormal Termination Processing)
  - APA (Automatic Profiling Analysis)
- Assessing Productivity Improvements
Productivity Background

- **The problem:** Large-scale scientific computers are getting larger and faster, but also more complex and more difficult to use
  - Complexity is especially challenging to new users
- HPCS Phase III Program specifically calls for improvements in developer productivity
  - This is completely separate from hardware performance improvements
  - Embodied in a set of 5 workflows. Developer productivity comes into play in 3 of them:
    - Writing large (multi-module) codes
    - Writing small codes
    - Porting codes
Level 1 Functional Workflows

(1) Writing Large Multi-Module Codes
- Formulate questions
- Develop Approach

(2) Writing Small Codes
- Develop Code
- V&V

(3) Running Codes
- Production Runs
- Analyze Results

(4) Porting Code
- Identify Differences
- Change Code
- Optimize

(5) Administration
- Problem Resolution
- Resource Management
- Security Management
- HW/SW Upgrade

- Workflows comprise many steps; many overlapping
- Item in red represent areas with highest HPC specific interest
Productivity Feature Work

- Cray is implementing a variety of new software and hardware features aimed at improving productivity
  - System Administration
    - identifying problems
    - upgrading system software
  - Writing new codes
    - Chapel language
      - “global-view” language, designed for parallel programming
      - See chapel.cray.com for more information
  - Compiling, Optimization and Debugging
    - Many features…. Luiz’s talk will cover this
    - Includes ATP and APA
Feature Assessments and Workflows

- Assess individual features or tools for their contribution to improving developer productivity
  - Compare how much time/effort when using the tool or feature vs. what effort was involved in the 2002 timeframe
- Will apply those improvements towards the workflows
- Starting with evaluations of 2 features:
  - ATP – a debug tool
  - APA – a feature of the performance analysis tool (CrayPat)

<table>
<thead>
<tr>
<th>Workflow 4: Porting</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Section</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Identify Differences</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Change Code</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Scale and Optimize</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Simplified Example of a Porting Workflow
ATP – Abnormal Termination Processing

- **The Problem**: When a parallel application dies, it is next to impossible to examine all the core files and backtraces
  - Core files
    - A single core file is usually not enough to debug
    - Sufficient storage for all core files is a problem
  - Backtraces
    - A single backtrace is usually not enough
    - The backtrace produced might not be from the process that first failed
    - Today’s systems produce one or none

- ATP produces a single merged stack trace or reduced set of core files. **The benefits:**
  - Easy to navigate the merged stack trace
  - Manageable set of core files
  - Reduced amount of data saved
    - Especially true in the core file situation
Simplified Workflow – Major Steps

Write Modify Port → Compile & Link → Execute (verification) → Debug → Optimize → Execute (production)

Iterate until you have a clean build
Iterate until you have a clean execution
Iterate until results are correct
Iterate until perf goals met

Compile & Link

Iterate until results are correct

Debug

Optimize

Execute (production)
Simplified Workflow with ATP

ATP operates during program execution

Write
Modify
Port

Compile & Link

Execute (verification)

Debug

Optimize

Execute (production)
ATP – Abnormal Termination Processing

Write
Modify
Port

Compile & Link

App runs (verification)
Exit

Normal Termination

Debug
Optimize

Abnormal Termination

ATP
Stacktrace (atpMergedBT.dot)
STATview

App runs (production)
Exit

Abnormal Termination

ATP
Stacktrace (atpMergedBT.dot)
STATview
ATP – How It Works

- ATP signal handler runs within an application. Its job is to catch fatal errors. It handles the following signals:
  - SIGQUIT, SIGILL, SIGTRAP, SIGABRT, SIGFPE, SIGBUS, SIGSEGV, SIGSYS, SIGXCPU, SIGXFSZ
  - Setting the environment variables MPICH_ABORT_ON_ERROR and SHMEM_ABORT_ON_ERROR will cause a signal to be thrown and captured for MPI and SHMEM fatal errors

- ATP daemon running on the compute node captures signals, starts termination processing
  - Rest of the application processes are notified
  - Generates a stacktrace
  - Creates a file named *.dot

- The *.dot file is viewed with the STATview tool
  - Pre-release of STATview is available on workshop systems
STATview Example
ATP – Future Features

- Automatic invocation of ATP
  - Today users need to insert signal handler
  - With next release of OS, just need to load atp module

- Core file subset
  - Intelligence from stack backtrace help decides which core files to produce

- Hold a dying application in stasis
  - Gives the user an opportunity to attach a debugger to the application

- Send email notification to user that job has failed

- Improved scalability
  - ATP stack backtraces have been produced on applications made up of about 2000 processes
  - Expect to be able to handle applications with 100,000s of processes in the future
ATP – Getting Started

- Get atp_example.tar from the Workshop website
  
  ```
  $ wget http://www.nersc.gov/projects/workshops/CrayXT/tbd
  $ tar -xvf atp_example.tar
  ```

- On a Cray XT with atp installed, type:
  
  ```
  $ module load xt-atp
  $ module load stat
  $ man intro_atp
  ```
APA – Automatic Profile Analysis

- **The Problem:** performance tools have many options and it can be a lot of work to set up options to profile a program with minimum overhead

- APA is an option that automatically creates a template file that can be used to set up a performance profile of the run

- **The Benefits:**
  - You can quickly and efficiently generate a performance profile
    - Automatically excludes those routines which took a small amount of time to reduce runtime overhead
    - Automatically specifies hardware counter groups
    - Automatically lists which libraries to profile
  - You do not need to wade through pages of documentation in order to do this
  - The template (.apa) file can subsequently be modified to refine the performance data collection
    - Also serves as usage documentation
Simplified Workflow with APA

1. Write
2. Modify
3. Port
4. Compile & Link
5. Execute (verification)
6. Debug
7. Optimize
8. Execute (production)
### APA – How It Works

1. **myprog**
   - **pat_build**
     - `-O apa`
   - **myprog+pat**
   - **aprun**
     - (execute)
   - **.xf file**
   - **pat_report**
   - **Performance report**

2. **pat_build**
   - `-O filename.apa`
   - **myprog+apa**
   - **aprun**
     - (execute)
   - **.xf file**
   - **pat_report**
   - **Performance report**

User can modify this template file

User analyzes this report
myprog

pat_build
-O apa

myprog+pat

aprun
(execute)

.xf file

pat_report

Performance report

pat_build
-O filename.apa

myprog+apa

aprun
(execute)

.xf file

pat_report

Performance report

Modify .apa

Performance report

Modify .apa

Optimize

2/02/2010 Cray Inc. Proprietary – Not For Public Disclosure
APA – How It Works

- User first instruments code with `pat_build -O apa`
  - Straightforward and requires little overhead when running
- User executes the application
  - The information needed to make a profile run is generated and produced in a file with the extension `.apa`
- Reinstrument the code (using `.apa` file)
- Rerun the code (produces `.xf` file)
- Produce the profile report
APA – Getting Started

- Get apa_example.tar from the Workshop website
  
  
  ```
  $ wget http://www.nersc.gov/projects/workshops/CrayXT/tbd
  $ tar -xvf apa_example.tar
  ```

- Alternatively:
  
  - See Section 2.4 *Using Automatic Program Analysis* in the manual
    *Using Cray Performance Analysis Tools* S-2376-50
  - Available on the docs.cray.com website

- Another alternative:
  
  ```
  $ module load xt-crøypat
  $ man intro_crøypat
  ```
Feature Assessments

- Objective is to answer the following questions:
  - *Does this feature help boost the productivity of developers?*
  - *How much does it help?*
  - *How easy was it to learn how to use the feature?*

- We are asking users to try out these features and report back on their experience.

- We are providing:
  - Quick, get-started guide for each feature which includes
    - Feature description
    - Feature benefit
    - How to
  - Simple example
    - Includes a shell script which walks through the steps
Feedback

- **How and when**
  - Fill in provided feedback forms during workshop
  - Talk to us during Hands-on time
  - Contact us via email
    - Margaret Cahir  n13671@cray.com
    - Don Mason  dmm@cray.com

- Would like to gather initial impressions of new tools and features
  - How easy it was to learn
  - How useful will it be
  - Time spent is of interest