Auto-Tuning Activities in Cray Scientific Libraries

Keita Teranishi
Cray Inc.
Challenges in industrial software/library development
CScADS 2009
YES!

- Cray has been developing auto-tuning framework to improve the productivity of scientific library development.
## generic ATF application

<table>
<thead>
<tr>
<th>Input</th>
<th>Compile</th>
<th>Search</th>
<th>Code Generation</th>
<th>Batch</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Define input syntax</td>
<td>• Compiles the executable</td>
<td>• Traverses the Search Space</td>
<td>• Defines loop transformation syntax</td>
<td>• Interacts with scheduler</td>
<td>• Performance analysis</td>
</tr>
<tr>
<td>• Builds I/O Classes</td>
<td>• Performs iterative compilation</td>
<td>• Greedy, random, orthogonal, genetic</td>
<td>• Builds multiple code variants</td>
<td>• Deduces concurrency level for search</td>
<td>• Decides when to stop</td>
</tr>
<tr>
<td>• Imports the Application configuration</td>
<td></td>
<td>algorithms</td>
<td>• Builds explicit code from generated</td>
<td>• Submits jobs</td>
<td>• Reports to temporary files and archives</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>algorithm (FFT)</td>
<td></td>
<td>• Writes end tables</td>
</tr>
</tbody>
</table>
Our Auto Tuning Philosophy

- We tune the libraries for the particular architectures (XT systems)
  - Adaptation to multiplatform is not important
    - Only happens within our platform
  - The size of search space is reasonable

- We prefer compile/installation time (off-line) auto-tuning to run-time tuning.
  - We have enough resources to take advantage of off-line auto-tuning

- We take advantage of the result of off-line tuning to eliminate the need of run-time tuning
  - The results of off-line tuning enable run-time (quick) adaptation
Our Auto Tuning Emphasis

- Use high level scripting language to steer all tuning processes
- Execution environment (such as batch environment) is very important
  - Running/killing many jobs (for speed of auto-tuning)
  - Recovery mechanism to rerun huge jobs (such as HPL)
  - Serves hardware sanity test
- Multiple code generator support
- Any tuning elements are parameterized
  - Loop transformation (annotation/directive based)
  - Input parameter of the library routines
  - Programming model (threaded/MPI/Hybrid)
  - Communication algorithms
  - Performance tuning environment variables
  - Compiler flags and type of compilers (GCC, PGI, Intel and Cray)
Tuning Sequences

- **FFT**
  - Input → CG → Batch → Search → Result

- **CrayHPL**
  - Input → Compile → Search → Batch → Result

- **CASK**
  - Input → CG → Compile → Search → Batch → Result
Lessons from Cray auto-tuning

- Auto-tuners need to come up with a reasonable performance model with a reasonable number of search parameters.

- Tuning for a particular problem size may not help the tuning for the other problem sizes
  - We need to find ways to model the performance for any problem sizes
  - HPL auto-tuning is a good example why the best parameters for small problems won’t work for large problems

- Auto-tuning improves the productivity of the performance tuning at the cost of tool development/integration
  - Integration is not straightforward though we have good tools (aprun, craypat, etc.)
  - Auto-tuner must have the same level of knowledge as hand-tuners do
What to be done

- Tuning for inter-node parallel applications
  - MPI, Shmem, PGAS
- Good code generators
  - Compiler with special directives
  - Code generator that produces assembly language for specific platform
  - Source-to-source transformation beyond loops for affined array access
- Good performance analysis tools and schemes
  - Necessary to establish performance model
  - Automatically find busy spot of the application code
- Automatic batch management system with fault recovery
- Easy integration with existing scientific libraries (or application code)
- Please make your tools portable to Cray platform!!