Towards Effective Productivity

The case for in-situ tracking of performance portability development

Authors:
The Three Ps: Performance, Portability, and Productivity

– Performance - Applications will need to run fast
  • The power wall prevents us from getting naive speed ups from new hardware
  • ...so we have to rely on smart programmers instead

– Portability - Applications need to be run on a wide range of systems, including ones that don’t exist yet
  • Expensive because of changing hardware architectures
  • Different architectures are optimised for different approaches to computation

– Productivity - Developers are needed to accomplish performance portability
  • Fast code costs developer time
  • Porting then maintaining multiple code paths for different architectures is expensive

Takeaway

We need to start accounting for all aspects of application efficiency in order to fully capture resource cost of an application. We will focus on productivity.
Parallel Computing Summer Research Internship at ISTI LANL

We will use productivity data and experiences collected from performance portability efforts during the internship for the following applications:

- VPIC - Conversion to PP Framework (Kokkos)
  - A first-principles particle-in-cell (PIC) plasma physics code

- Truchas - Performance portability study with a variety of CPU and GPU frameworks
  - A 3D multi-physics simulation tool for metal casting and other applications

- Spectral BTE - Optimization by refactoring and offloading to GPU
  - Used to model molecules that are not in equilibrium and has applications in hypersonic flows, fluid micro-flows, plasma physics

Program Leads: Robert (Bob) Robey, Hai Ah Nam, Kris Garrett, Eunmo Koo, Luke Van Roekel
Measure for Performance Portability

Performance Portability

\[
PP = \begin{cases} 
\frac{1}{\sum_{p \in H} \frac{1}{e(a, p)}} & \text{if all platforms are supported} \\
0 & \text{if any platform is unsupported}
\end{cases}
\]

On a given set of platforms, the performance portability of an application is the harmonic mean of the performance efficiencies on each platform or 0 if any platform is unsupported [1].

Methodology to Capture Performance Metrics

- Well known and application specific
### Performance Portability Results for VPIC and Truchas

**TABLE II: Application efficiency of the `advance_b` and `advance_p` Kernels in VPIC**

<table>
<thead>
<tr>
<th>Platform</th>
<th><code>advance_b</code></th>
<th></th>
<th><code>advance_p</code></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>Kokkos</td>
<td>Original</td>
<td>Kokkos</td>
</tr>
<tr>
<td>Intel® Xeon® 8176 Processor</td>
<td>100%</td>
<td>13%</td>
<td>100%</td>
<td>62%</td>
</tr>
<tr>
<td>IBM® Power 9</td>
<td>100%</td>
<td>32%</td>
<td>100%</td>
<td>46%</td>
</tr>
<tr>
<td>Cavium® ThunderX2</td>
<td>100%</td>
<td>54%</td>
<td>100%</td>
<td>50%</td>
</tr>
<tr>
<td>NVIDIA® V100</td>
<td>0%</td>
<td>100%</td>
<td>0%</td>
<td>100%</td>
</tr>
</tbody>
</table>

**TABLE III: Architectural efficiency of the `advance_b` and `advance_p` Kernels in VPIC**

<table>
<thead>
<tr>
<th>Platform</th>
<th><code>advance_b</code></th>
<th></th>
<th><code>advance_p</code></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original</td>
<td>Kokkos</td>
<td>Original</td>
<td>Kokkos</td>
</tr>
<tr>
<td>Intel® Xeon® 8176 Processor</td>
<td>12%</td>
<td>2%</td>
<td>18%</td>
<td>11%</td>
</tr>
<tr>
<td>IBM® Power 9</td>
<td>14%</td>
<td>5%</td>
<td>8%</td>
<td>4%</td>
</tr>
<tr>
<td>Cavium® ThunderX2</td>
<td>11%</td>
<td>6%</td>
<td>10%</td>
<td>5%</td>
</tr>
<tr>
<td>NVIDIA® V100</td>
<td>0%</td>
<td>93%</td>
<td>0%</td>
<td>5%</td>
</tr>
</tbody>
</table>

**TABLE VII: Performance portability based on architectural efficiency for the Mimetic Finite Difference Kernel**

<table>
<thead>
<tr>
<th>Platform</th>
<th>Version</th>
<th>Arch. Eff.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Xeon® E5-2698 processor</td>
<td>OpenMP CPU</td>
<td>4.19%</td>
</tr>
<tr>
<td>Intel® Xeon Phi™ 7250 processor</td>
<td>OpenMP CPU</td>
<td>7.76%</td>
</tr>
<tr>
<td>Intel® Xeon® Platinum 8176 processor</td>
<td>OpenMP CPU</td>
<td>5.49%</td>
</tr>
<tr>
<td></td>
<td>Power9</td>
<td>3.44%</td>
</tr>
<tr>
<td></td>
<td>Volta + Power9</td>
<td>5.41%</td>
</tr>
<tr>
<td></td>
<td>Volta + Power9</td>
<td>CUDA</td>
</tr>
<tr>
<td>Volta + Intel® Xeon® E5-2683 processor</td>
<td>CUDA</td>
<td>90.04%</td>
</tr>
<tr>
<td>Performance Portability</td>
<td></td>
<td>6.7%</td>
</tr>
</tbody>
</table>

Measured productivity of these projects
Prior Productivity Tracking Efforts

Prior efforts include:

• SLOCCount
  ○ Uses the Source Lines of Code to estimate developer effort

• Function Points
  ○ A unit of measurement which captures the amount of functionality provided to a user
  ○ Harder to define for scientific codes

• COCOMO II
  ○ COnstructive COst MOdel II
  ○ “A model that allows one to estimate the cost, effort, and schedule when planning a new software development activity” [6]
Code Divergence

\[ D(A) = \left( \frac{|A|}{2} \right)^{-1} \sum_{\{a_i, a_j\} \subset A} d(a_i, a_j) \] (3)

- **Pairwise distance between two codebases** (d)
  - Can use different types of functions for (d)
  - Method we use is a differencing two code paths in one code base and normalise based on the size of the total code base

- **Combine pairwise distance metrics to determine code divergence** (3)

- **Intuitively we would expect that the maintenance cost of any code bases increases proportionally to the divergence** (D)

- **VPIC - Compared the GPU vs CPU codepaths using the kokkos port of VPIC**
  - 2 line difference, 18198 total lines
  - 0.01% divergence - expected maintenance cost being very close to the cost of a single code path
In-Situ Methodology to Capture Productivity Metrics

Developers code like normal
- After making a commit, they answer a few extra questions about their work
- Their answers, plus some metadata, are stored as hidden files (git notes) on each commit
In-Situ Methodology to Capture Productivity Metrics

Developers code like normal
- After making a commit, they answer a few extra questions about their work
- Their answers, plus some metadata, are stored as hidden files (git notes) on each commit

Post-processing captures logging info
- Extracts commit graph
- Copies over hidden log files
- Logs are machine-readable (JSON); can be trawled for info and analysed easily

Motivation for Mirror Repo
The code and productivity log are not coupled so one can be used without the other in case of encumbered data
Example Productivity Results

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLOC Changed</td>
<td>0.48</td>
</tr>
<tr>
<td>Coding Time</td>
<td>0.46</td>
</tr>
<tr>
<td>Coding Difficulty</td>
<td>0.45</td>
</tr>
<tr>
<td>Debugging Time</td>
<td>0.47</td>
</tr>
<tr>
<td>Debugging Difficulty</td>
<td>0.71</td>
</tr>
<tr>
<td>Planning Time</td>
<td>0.32</td>
</tr>
<tr>
<td>Planning Difficulty</td>
<td>0.39</td>
</tr>
<tr>
<td>Refactoring Time</td>
<td>0.44</td>
</tr>
<tr>
<td>Refactoring Difficulty</td>
<td>0.33</td>
</tr>
<tr>
<td>TLX Effort</td>
<td>0.39</td>
</tr>
<tr>
<td>TLX Frustration</td>
<td>0.22</td>
</tr>
<tr>
<td>TLX Mental Demand</td>
<td>0.86</td>
</tr>
<tr>
<td>TLX Performance</td>
<td>0.37</td>
</tr>
<tr>
<td>TLX Temporal Demand</td>
<td>0.16</td>
</tr>
<tr>
<td>Commit Includes Cuda</td>
<td>0.01</td>
</tr>
<tr>
<td>Commit Includes Kokkos</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Los Alamos National Laboratory
Example Productivity Results

Development Progress
Example Productivity Results

**Mean: 1.324983 minutes per line**

- **Time (Hours)**
  - X-axis: 0 to 25
  - Y-axis: 0 to 600

**Mean: 2.181700 difficulty per 100 lines**

- **Difficulty of Total Commit**
  - X-axis: 1 to 7
  - Y-axis: 0 to 600

**Difficulty per Line vs Total Time Spent**

- **Difficulty per Line of Code Changed**
  - X-axis: 0 to 20
  - Y-axis: 0 to 0.4

**Self-Reported Difficulty**

- **Planning**: 5.14
- **Coding**: 3.62
- **Refactoring**: 3.22
- **Debugging**: 4.52

Los Alamos National Laboratory
Example Productivity Results

How Development Time was Spent

Los Alamos National Laboratory
Conclusions

Performance Portability

● VPIC
  ○ Using Kokkos the VPIC team was able to convert half of the kernels in VPIC in a very short amount of time.
  ○ With this method code divergence was very low. Once completely ported the costs of maintenance and ports to new architectures are expected to be low.

● Truchas
  ○ Discussed later in this workshop in the “Performance Portability Challenges for Fortran Applications” talk

Productivity

● We developed productivity logging tools
  ○ Minimally intrusive/didn’t impede development significantly
  ○ Data yielded time series tied to project state

● Proposed Metrics for code divergence and its link to maintenance cost
Future Work

- Release Git Productivity Logging Tools

- **We need more data about productivity**
  - More projects
  - Larger teams
  - Hackathons and code sprints

- **Explore the connection between maintenance cost and code divergence**
We would like to thank Dr. Hai Ah Nam, and Dr. Kris Garrett for their help and advice over the course of the summer. The Darwin cluster at LANL was used for this work.