Towards HPRC Application Programming in C

David Pointer
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
Presentation outline

• Introduction
• Application: MATPHOT
• Application: BLAST
• Application: NAMD
• Conclusions
Introduction
Innovative Systems Lab

• Volodymyr Kindratenko
  <kindr@ncsa.uiuc.edu>
• Dave Raila <raila@cs.uiuc.edu>
• Craig Steffen <csteffen@ncsa.uiuc.edu>
Introduction

Innovative Systems Lab

- Cray XD1 / DSPlogic / Handel C
- SGI Altix 350/MOATB RASC / Handel C
- SGI Altix 350/MOATB RASC / Mitrion C
- SRC MAPstation / native C tools
Introduction
Innovative Systems Lab

- ISL Mission - Be a bridge between the early adopter application scientists and machines that have some performance potential beyond Moore’s Law.
- As much as possible, let the application scientists do science rather than programming and hardware engineering.
ISL Reconfigurable Computing
The Plan

• Develop rules of thumb – RC programming idiom.
• No VHDL, Verilog, or hardware to learn.
• Work with application scientists to generate performance improvements to real world floating point code.
• Eventually, teach application scientists HPRC programming.
MATPHOT

• Author
  – Kenneth Mighell, National Optical Astronomy Observatory

• What it does
  – Algorithm for Accurate and Precise Stellar Photometry and Astrometry Using Discrete Point Spread (PSF) Functions

• Where to get it
  – http://www.noao.edu/staff/mighell/matphot/
MATPHOT

• Simulated observational data

• Best model of the observation

Images are courtesy of Kenneth Mighell from the National Optical Astronomy Observatory

National Center for Supercomputing Applications
## MATPHOT Code Profiling

<table>
<thead>
<tr>
<th>% time</th>
<th>cumulative seconds</th>
<th>self seconds</th>
<th>calls</th>
<th>self s/call</th>
<th>total s/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>84.17</td>
<td>82.39</td>
<td>82.39</td>
<td>91504</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7.49</td>
<td>89.72</td>
<td>7.33</td>
<td>152</td>
<td>0.05</td>
<td>0.59</td>
<td>mpd_iplimageS_sshift_fnV5_VK</td>
</tr>
<tr>
<td>5.39</td>
<td>95</td>
<td>5.28</td>
<td>152</td>
<td>0.03</td>
<td>0.63</td>
<td>mpd_fdels0d_fs7</td>
</tr>
<tr>
<td>0.87</td>
<td>95.85</td>
<td>0.85</td>
<td>65</td>
<td>0.01</td>
<td>0.01</td>
<td>mpd_LmmS_AlphaBetaSet_fs1</td>
</tr>
<tr>
<td>0.5</td>
<td>96.34</td>
<td>0.49</td>
<td>152</td>
<td>0</td>
<td>0</td>
<td>mpd_iplimageS_Add_star_fs7</td>
</tr>
<tr>
<td>0.3</td>
<td>96.63</td>
<td>0.29</td>
<td>142</td>
<td>0</td>
<td>0</td>
<td>mpd_fmodel_fs7</td>
</tr>
<tr>
<td>0.15</td>
<td>96.78</td>
<td>0.15</td>
<td></td>
<td></td>
<td></td>
<td>snorm</td>
</tr>
</tbody>
</table>
MATPHOT Code Performance - I

Image size (pixels)

- CPU performance
- MAP performance

Compute time (s)

- Speed-up factor

128 256 512 1024 2048 4096 8192

0.1 0.2 0.5 0.9 1.5 2.1 2.3

National Center for Supercomputing Applications
MATPHOT Code Performance - I

![Graph showing compute time vs image size]

- **I/O only**
- **I/O+compute**

Image size (pixels):
- 128
- 256
- 512
- 1024
- 2048
- 4096

Compute time (s): 0 to 7

*National Center for Supercomputing Applications*
MATPHOT Code Performance - II

- Why is that?

<table>
<thead>
<tr>
<th>% time</th>
<th>cumulative seconds</th>
<th>self seconds</th>
<th>calls</th>
<th>self s/call</th>
<th>total s/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>84.17</td>
<td>82.39</td>
<td>82.39</td>
<td>91504</td>
<td>0</td>
<td>0</td>
<td>mpd_ipImageS_sshift_fnV5_VK</td>
</tr>
<tr>
<td>7.49</td>
<td>89.72</td>
<td>7.33</td>
<td>152</td>
<td>0.05</td>
<td>0.59</td>
<td>mpd_ipImageS_Shift2d_fs4</td>
</tr>
<tr>
<td>5.39</td>
<td>95</td>
<td>5.28</td>
<td>152</td>
<td>0.03</td>
<td>0.63</td>
<td>mpd_fdel0d_fs7</td>
</tr>
<tr>
<td>0.87</td>
<td>95.85</td>
<td>0.85</td>
<td>65</td>
<td>0.01</td>
<td>0.01</td>
<td>mpd_LmmS_AlphaBetaSet_fs1</td>
</tr>
<tr>
<td>0.5</td>
<td>96.34</td>
<td>0.49</td>
<td>152</td>
<td>0</td>
<td>0</td>
<td>mpd_ipImageS_Add_star_fs7</td>
</tr>
<tr>
<td>0.3</td>
<td>96.63</td>
<td>0.29</td>
<td>142</td>
<td>0</td>
<td>0</td>
<td>mpd_fmodel_fs7</td>
</tr>
<tr>
<td>0.15</td>
<td>96.78</td>
<td>0.15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>snorm</td>
</tr>
</tbody>
</table>

- So, which function should we port?
  - mpd_ipImageS_Shift2d_fs4 (proc_cpu)
MATPHOT Code Performance - II

The diagram shows a comparison of compute time (s) for different image sizes (pixels) between CPU and MAP. The speed-up factor is indicated for each data point. The image size is shown on the x-axis, ranging from 128 to 4096 pixels. The compute time for CPU is represented by red squares, and for MAP by green squares. The speed-up factor is indicated by the blue shaded area. The graph illustrates that MAP outperforms CPU significantly, especially as the image size increases.
MATPHOT Code Performance - II

![Graph showing compute vs I/O performance comparison chart](image.png)
MATPHOT Lessons Learned

• Port algorithm, not code
• Beware Memory I/O Overhead
  – Check function call frequency
  – Call FPGA empty function
  – Call FPGA with data transfer only
  – Add real design to FPGA
• Leave CPU assumptions behind
  – 2D convolution faster than 1D convolution
BLAST

• Is it possible that a given protein could be generated by a given DNA sequence?
• Matt Hudson, Department of Crop Science, UIUC
• Working with an example plant protein and DNA sequence for test case.
BLAST – Algorithm

30M bases

1200 amino acids
BLAST – Lessons Learned (so far)

• Sometimes you need to restructure your code to work around compiler technology limitations.
NAMD

• Simulate atom interactions based on distance and atomic forces
• Jim Phillips, Beckman Institute, UIUC
• Working on second pass implementation of code, not algorithm.
NAMD - Algorithm

• Each atom represented by 7 SPFP and 1 integer values.
• Code inner loop – calculate force equation for each of 100,000 atoms to 1 atom.
• Code outer loop – presort list and call inner loop for each of suitable 100,000 atoms.
NAMD – Motivation

• Single largest cycle user of NCSA production machines.
• Currently can simulate a single femtosecond simulation step in 4 milliseconds on a 16 node cluster, 10 milliseconds on a single CPU.
• If we can get down do 1 millisecond per femtosecond step, Jim Phillips’ group could do simulated real time molecule building by human hand.
NAMD – First Implementation

• Inner loop implemented in FPGA.
• 200X slowdown.
NAMD – Lessons Learned (so far)

• Empty fpga function call yields 50x slowdown. See MAPHOT lesson on function call frequency.
• Well, maybe you should port code instead of the algorithm if there are a lot of excellent optimizations embedded in the code – order of magnitude optimization rules in outer loop.
Conclusions

• RC programming is not for the weak of heart
  – Be ready to throw out years of work
  – Be ready to question assumptions
  – Be ready to throw out what you know to be true

• We are still learning
Questions?

David Pointer <pointer@ncsa.uiuc.edu>