Benchmarking in AIPS++

Sanjay Bhatnagar
NRAO, Socorro
Approach to benchmarking

● Develop easy-to-run tools

  Populate the AIPS++ Benchmark module

  Make same and repeatable measurements

● Develop conceptual understanding of the compute, I/O and other major costs of the algorithms

● Reduce (eliminate) extraneous parameters

● Understand performance curves (and not spot measurements) as a function of the problem size

● Start from the simplest, but realistic problem size
Work so far...

- **Image re-gridder optimization**
  - Low level optimization (Brouw)
  - Achieved a factor of ~10 improvement. Real-time image re-gridding possible

- **Interferometric imaging and calibration:** *Study of the algorithms and the associated I/O costs*

- **Equivalent scripts for other popular packages (AIPS, Miriad)**

- **Profiling:** Very useful and in progress using standard GNU tools

- **Regular profiling builds of AIPS++**

- **Dedicated computer for benchmarking**
  - Need isolation and controlled environment
Work so far...

- **Developed benchmark**
  Control on algorithms, problem size, data size, Table system parameters, memory model...

- **ALMA data size tests**
  - First cut at ALMA TI filler (Rusk)
  - Iramcalibrator compared with CLIC.

- **Well known bottle-necks being optimized**

- **Profiling and optimization in progress**
Results: General

- **AIPS performance curves were taken as the 'baseline'** *(Miriad is 2x faster than AIPS though!)*

- **AIPS++ default settings too conservative**
  
  - Memory/CPU usage monitoring indicated memory model too conservative for present day computers
  
  - Default settings can be (should be?) made computer-resource-aware

- **Burnt-in assumptions for memory model in some critical parts of the code**

- **Redundant (expensive) operations**

- **Profiling ultimately dug out a few real & 'dormant bugs'**
Results: Well-known mistakes

- **Tiling** in the construction of Tables is an important parameter
- **Array access pattern**: can be expensive if done the wrong way
- **Inadvertent use of copy vs. reference semantics** is expensive
  - Results into excessive memory copies
  - Use of objcopy() should be minimized
- **Table creation may be expensive**
  - Used by ArrayLattice in tiling mode
  - Minimize Table creation inside tight loops
- **Strictly use only OSInfo() class for memory model**
Results: ALMA Tests

- Performance studied as a function of data-rate (Rusk)
- Imaging appears to scale well
- Hardware/compilers factor in as significant parameters
- Understand the scaling as a function of various parameters
- More work needed to understand the differences in these curves
Results: IRAM Calibrator Tests

- **First cut at iramcalibrator benchmarking**
  (Rusk/Lucas/Golap)
  
  No. of antennas: 64

<table>
<thead>
<tr>
<th>Calib. cycle</th>
<th>AIPS++ (sec)</th>
<th>AIPS++ (sec)</th>
<th>CLIC (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF</td>
<td>2646</td>
<td>1818</td>
<td>2187</td>
</tr>
<tr>
<td>Phase</td>
<td>1404</td>
<td>804</td>
<td>942</td>
</tr>
<tr>
<td>Amp</td>
<td>2502</td>
<td>2076</td>
<td>708</td>
</tr>
</tbody>
</table>

- **AIPS++ overheads insignificant**
- **AIPS++ code not yet fully optimized**
- **RF & Phase solvers use the same engine in AIPS++ and CLIC: runtime within 10-20% of each other**
- **Amp: same solver, but AIPS++ includes apply()**
Results: Imager

- **Clark-CLEAN algorithm** (one of the many in AIPS++)

- **Recognize 'minor' /subtle differences in the implementation**
  - AIPS uses Cotton-Schwab variant

- **FFTW**: optimize data locality and in-memory copies

- **Threaded FFT**: useful on ubiquitous multi-CPU machines
Results: Calibrater

- Gain and band-pass calibration was not optimal (fixed)
- Slot hunting algorithm was non-optimal (fixed)
- Profile dominated by lower level calls
- More work needed once these major bottlenecks are removed

Gain and polarization leakage calibration comparable to AIPS till ~200 solution intervals.
Results: I/O test (not well understood)

- **Measure I/O costs as seen by the AIPS++ 'algorithm layer'**
- **Table system's cache-hit rate was >99%**
- **Need more work**

![Graph showing runtime vs. PSF size (pixels)](image-url)
Lessons learned

- *Benchmarks are a measure in a multi-dimensional space*

  - Requires careful analysis and understanding of the software and the hardware platform
  - Various axis are not often orthogonal

- *Benchmarking and optimization is better done closer to code development than much later.*

- *Profiling is the single most useful tool.*

- *No fundamental hot-spots (IMHO!)*
...continued

• **AIPS++ framework is highly configurable**
  - Memory model is user configurable
  - Table I/O is configurable (need to bring it to the user level?)

• **Smarter automatic (computer resource-aware) settings is an enticing possibility**

• **Better Technical documentation**
  - Cost analysis for developers
  - Do's and Don't's for developers