Coding Tools

(Lectures on High-performance Computing for Economists VI)

Jesús Fernández-Villaverde,1 Pablo Guerrón,2 and David Zarruk Valencia3
September 14, 2020

1University of Pennsylvania
2Boston College
3ITAM
Compilers
If you use a compiled language such as C/C++ or Fortran, you have another choice: which compiler to use?

**Huge differences among compilers in:**

1. Performance.
2. Compatibility with standards.
3. Implementation of new features:
   
4. Extra functionality (MPI, OpenMP, CUDA, OpenACC, ...).

**High return in learning how to use your compiler proficiently.**

**Often you can mix compilers in one project.**
## Linux/64 on Intel Processor

<table>
<thead>
<tr>
<th></th>
<th>Absoft</th>
<th>Absoft(AP)</th>
<th>gfortran</th>
<th>Intel</th>
<th>Intel(AP)</th>
<th>NAG</th>
<th>Oracle</th>
<th>PGI</th>
<th>open64</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>5.01</td>
<td>4.96</td>
<td>6.36</td>
<td>4.47</td>
<td>4.44</td>
<td>7.32</td>
<td>18.91</td>
<td>6.84</td>
<td>5.07</td>
</tr>
<tr>
<td>AERMOD</td>
<td>11.35</td>
<td>11.51</td>
<td>15.60</td>
<td>11.52</td>
<td>12.26</td>
<td>18.64</td>
<td>11.13</td>
<td>11.98</td>
<td>15.79</td>
</tr>
<tr>
<td>AIR</td>
<td>3.50</td>
<td>2.18</td>
<td>3.16</td>
<td>2.60</td>
<td>1.98</td>
<td>4.94</td>
<td>2.78</td>
<td>3.32</td>
<td>3.46</td>
</tr>
<tr>
<td>CHANNEL2</td>
<td>73.53</td>
<td>28.58</td>
<td>83.00</td>
<td>84.76</td>
<td>28.95</td>
<td>105.87</td>
<td>84.26</td>
<td>81.57</td>
<td>103.80</td>
</tr>
<tr>
<td>DODUC</td>
<td>19.28</td>
<td>19.34</td>
<td>18.70</td>
<td>15.12</td>
<td>14.97</td>
<td>24.20</td>
<td>15.96</td>
<td>18.11</td>
<td>18.72</td>
</tr>
<tr>
<td>FATIGUE2</td>
<td>63.09</td>
<td>66.65</td>
<td>67.08</td>
<td>55.62</td>
<td>55.75</td>
<td>117.37</td>
<td>82.94</td>
<td>89.12</td>
<td>77.66</td>
</tr>
<tr>
<td>GAS_DYN2</td>
<td>73.96</td>
<td>49.13</td>
<td>86.23</td>
<td>62.25</td>
<td>38.42</td>
<td>177.37</td>
<td>74.91</td>
<td>111.19</td>
<td>79.02</td>
</tr>
<tr>
<td>INDUCT2</td>
<td>83.68</td>
<td>76.03</td>
<td>80.99</td>
<td>71.82</td>
<td>50.96</td>
<td>132.20</td>
<td>138.92</td>
<td>127.38</td>
<td>144.11</td>
</tr>
<tr>
<td>LINPK</td>
<td>5.23</td>
<td>5.49</td>
<td>4.93</td>
<td>4.37</td>
<td>4.47</td>
<td>6.22</td>
<td>4.70</td>
<td>5.96</td>
<td>5.73</td>
</tr>
<tr>
<td>MDBX</td>
<td>9.68</td>
<td>7.98</td>
<td>8.07</td>
<td>6.47</td>
<td>4.85</td>
<td>8.68</td>
<td>8.54</td>
<td>9.08</td>
<td>9.35</td>
</tr>
<tr>
<td>MP_PROP_DESIGN</td>
<td>120.85</td>
<td>13.13</td>
<td>157.61</td>
<td>62.78</td>
<td>10.97</td>
<td>254.30</td>
<td>196.16</td>
<td>88.35</td>
<td>127.22</td>
</tr>
<tr>
<td>NF</td>
<td>8.13</td>
<td>8.21</td>
<td>7.30</td>
<td>7.58</td>
<td>7.54</td>
<td>9.00</td>
<td>8.98</td>
<td>8.47</td>
<td>7.96</td>
</tr>
<tr>
<td>RNFLOW</td>
<td>15.55</td>
<td>15.23</td>
<td>13.66</td>
<td>12.52</td>
<td>9.65</td>
<td>16.66</td>
<td>17.34</td>
<td>17.00</td>
<td>21.54</td>
</tr>
<tr>
<td>TEST_FPU2</td>
<td>61.24</td>
<td>43.00</td>
<td>50.15</td>
<td>43.44</td>
<td>39.64</td>
<td>82.37</td>
<td>64.90</td>
<td>48.28</td>
<td>57.10</td>
</tr>
<tr>
<td>TFFT2</td>
<td>58.66</td>
<td>61.10</td>
<td>46.74</td>
<td>58.95</td>
<td>62.43</td>
<td>60.41</td>
<td>58.91</td>
<td>56.78</td>
<td>58.55</td>
</tr>
<tr>
<td>Geometric Mean</td>
<td>22.93</td>
<td>17.39</td>
<td>22.95</td>
<td>19.33</td>
<td>14.96</td>
<td>30.95</td>
<td>26.44</td>
<td>24.23</td>
<td>25.45</td>
</tr>
</tbody>
</table>
The GCC compiler collection

- A good default option: GNU GCC 10.2 compiler.

1. Open source.


3. Integrates well with other tools, such as JetBrains’ IDEs.

4. Updated (C++20).

5. Efficient.

6. *An Introduction to GCC*, by Brian Gough,

1. LLVM (http://llvm.org/), including Clang.

   1.1 It comes with macOS and Xcode.

   1.2 Faster for compiling, uses less memory.

   1.3 Run time is slightly worse than GCC.

   1.4 Useful for extensions: Cling (https://github.com/root-project/cling).

   1.5 Architecture of Julia.

2. DragonEgg: uses LLVM as a GCC backend.

2. PGI. Community edition available. Good for OpenACC.


5. Fortran: Absoft, Lahey, and NAG.
Libraries
Libraries

• Why libraries?

• Well-tested, state-of-the-art algorithms.

• Save on time.

• Classic ones:

  1. BLAS (Basic Linear Algebra Subprograms).

  2. Lapack (Linear Algebra Package).
• More modern implementations:

1. Accelerate Framework (macOS).
2. ATLAS (Automatically Tuned Linear Algebra Software).
3. MKL (Math Kernel Library).

• Open source libraries:

1. GNU Scientific Library.
2. GNU Multiple Precision Arithmetic Library.
3. Armadillo.
4. Boost.
5. Eigen.
Build Automation
A build tool automatizes the linking and compilation of code.

This includes latex and pdf codes!

Why?

1. Avoid repetitive task.
2. Get all the complicated linking and compiling options right (and, if text, graphs, options, etc.).
3. Avoid errors.
4. Reproducibility.

GNU Make and CMake.
Why Make?

- Programed by Stuart Feldman, when he was a summer intern!
- Open source.
- Well documented.
- Close to Unix.
- Additional tools: etags, cscope, ctree.
Basic idea

- You build a make file: script file with:
  1. Instructions to make a file.
  2. Update dependencies.
  3. Clean old files.

- Daily builds. Continuous integration proposes even more.

Containers

- A container is stand-alone, executable package of some software.

- It should include everything needed to run it: code, system tools, system libraries, settings, ...

- Why? Keep all your environment together and allow for multi-platform development and team coding.

- Easier alternative to VMs. But dockers are not “lightweight VMs.”

- Most popular: Docker [https://www.docker.com/](https://www.docker.com/).

- Built around dockerfiles and layers.
Linting
Linting

- Lint was a particular program that flagged suspicious and non-portable constructs in C source code.
- Later, it became a generic word for any tool that discovers errors in a code (syntax, typos, incorrect uses) before the code is compiled (or run) ⇒ static code analyzer.
- It also enforces coding standards.
- Good practice: never submit anything to version control (or exit the text editor) unless your linting tool is satisfied.

Examples:

1. Good IDEs and GCC (and other compilers) have excellent linting tools.
2. C/C++: clang-tidy and ccmcheck.
4. R: lintr.
5. Matlab: checkcode in the editor.
Debugging
If you’re confident your code works, you’re probably wrong. And that should worry you.

- Find and eliminate mistakes in the code.
- In practice more time is spent debugging than in actual coding.
- Complicated by the interaction with optimization.
- Difference between a bug and a wrong algorithm.
8:00  Action started
1000 Stopped - action ↑

13°C (033) MP - MC  [530.4/22.5/25] 4.515925057 (2)

033 PRO = 2.130 966 15

Con'd 2.130 966 15

Relays 0-2 m 033 failed speed speed test
In Relay
Relays changed
1100  Started Cosine Tape (Sine check)

1525 Started Multi Adder Test.

1545 Relay #70 Panel F
(Moth) in relay.

1600  Antennas started.
1700  Sound down.
Typical bugs

- Memory overruns.
- Type errors.
- Logic errors.
- Loop errors.
- Conditional errors.
- Conversion errors.
- Allocation/deallocation errors.
How to avoid them

• Techniques of good coding.

• Error handling.

• Strategies of debugging:
  1. Tracing: line by line.
  2. Stepping: breakpoints and stepping over/stepping out commands.
  3. Variable watching.
Debuggers

- Manual inspection of the code. Particularly easy in interpreted languages and short scripts.

- Use assert.

- More powerful → debuggers:

  1. Built in your application: RStudio, Matlab or IDEs.

  2. Explicit debugger:

    2.1 GNU Debugger (GDB), installed in your Unix machine.

    2.2 Python: pdb.

    2.3 Julia: https://julialang.org/blog/2019/03/debuggers/.
Before going down a steep hill like this, one should always give his sled a safety check.

Right.

Seat belts? None.

Signals? None.

Brakes? None.

Steering? None.

Wheeeeee
Unit testing

- Idea.

- Tools:
  1. xUnit framework (CppUnit, testthat in R, ...).
  2. In Julia: Test module.

- Regression testing.
Profiler
Profiler

- You want to identify the hot spots of performance.

- Often, they are in places you do not suspect and small re-writings of the code bring large performance improvements.

- Technique:
  1. Sampling.
  2. Instrumentation mode.

- We will come back to code optimization.