State Of The Performance Tools (A Personal Opinion)

Douglas Pase, PhD (CSE)

Sandia National Laboratories/SAIC

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This unclassified document is approved for unlimited release
(**SAND2017-8098 C**).

Computing Environment At Sandia

- Demanding hardware environment
  - Multiple very large clusters and MPPs, variety of CPU and network architectures
- Demanding platform operation environment
  - Multiple operating systems, many compiler and MPI releases
- Demanding application environment
  - Many applications, long running, large code bases, multiple languages (as old as Fortran 77 to as new as C++17), multiple programming and concurrency models, multi-physics, multiple numerical techniques and always under development
  - Build (make/cmake) scripts are often large, complex, convoluted and opaque
- Most developers are Subject Matter Experts working under tight deadlines, often rewarded for features over performance, with little time or patience for the vagaries of cranky tools
- There is a support team who understands both tools and applications

How Do You Build Tools For *That*?

- Tools must be:
  - Easy to use – tools with many steps and many arguments **don't get used**
  - Robust – tools that break easily **don't get used**
  - Flexible – tools that don't work across our many environments **don't get used**
  - Lightweight – tools that demand many system resources **don't get used**
  - Scalable – tools that don't work at full scale, full optimization **don't get used**
  - Informative – tools that don't easily give useful information **don't get used**
- Documentation must include simple recipes for common problems, including as basic as "Hello, world!"

Swiss Army Knife Or Multiple Tools?

- Most often used tool at Sandia is custom instrumentation using gettimeofday(), rdtsc(), local counters and printf()
  - Light weight, understood by developers, robust, easy to use, gives measurements in terms of application concepts
  - Caliper is able to make this easier
- Single tool that does a few things well, e.g., Allinea MAP?
- Or many separate tools with variety of data, e.g., Open|Speedshop?
- There is a trade-off between simplicity and information
- Often useful to have metrics of a certain class (e.g., MPI, OMP, PMU) presented together, but not as important when they're unrelated

Easy To Use

- Easy to instrument applications and collect data
- Transparent analysis phase
- Easy to start the data browser
- Easy to navigate the data
- Easy to remember (consistent across installations)

Flexible And Robust

- Works reliably with a wide variety of hardware and software
  - Processor and network hardware
  - Operating systems
  - Languages, including mixed language programs
  - Compiler versions
  - Library selections

Lightweight

- Extra start-up and shut-down time is annoying but tolerated
- Time dilation in the measurements is not accepted
- Memory use must be kept small

Scalable

- Large applications (2.5+ million LOC)
- Large, complex libraries (e.g., Boost, Trilinos, Kokkos, Zoltan, MKL)
- Large numbers of cores (e.g., O(10,000) cores or larger)
- Large (wide and deep) networks

Informative

- Performance data must reflect production code and work flow
- Data must be gathered at full scale under full optimization
- Must be presented in terms familiar to the audience
  - Mapped to program scope (call chain, subroutine or lines of code)
  - Recognizable metrics (e.g., MPI wait time, OMP barrier time)
- Unfortunately, the *symptoms* may occur far from their *causes*
  - Barrier wait time may be far from the work distribution code that causes it
  - Choice of mesh upstream may impact downstream performance

Tool A - LDPXI

Pros

- Very light weight
- Variety of measurements – MPI, PAPI, I/O, BLAS, memory
- Easy to use
- Gives a summary of the program

Cons

- Home grown
- Does not work with OpenMP
- Struggles with older Fortran
- Requires dynamically loaded libraries
- Doesn’t multiplex PAPI counters (requires multiple data runs)

Tool B

Pros

- Very light CPU overhead
- Very easy to gather data
- Intuitive browser layout
- Finds symptoms quickly

Cons

- Occasional heavy memory usage (but being addressed)
- Each MPI rank requires a license (large runs run out of licenses)

Tool C

Pros

- Intuitive browser interface

Cons

- Separate steps to run, analyze and browse performance data
- Analysis fails on large codes
- User selection of sample rate (easy to get wrong)

Tool D

Pros

- Single purpose tool (MPI data)
- Easy to use (set an env. variable)
- Detailed data includes call chains to MPI call sites

Cons

- Output can exceed 500,000 lines of text on real codes and runs
- Information gets lost in the mass of data
- Needs a separate build for nearly every compiler/MPI combination

Tool E

Pros

- Wide range of data collection
- Consistent look and feel across all data browsing

Cons

- Many ways to accomplish the same task
- User must select some of the data collection parameters – error prone and fragile
- More robust on some clusters than others (work in progress)

Tool F

Pros

- Flexible instrumentation
  - Linked in dynamically
  - Compiled in
  - Added manually
- Extensive set of metrics
  - PMU/PAPI, MPI, mem., OMP, I/O
- Works with Vampir

Cons

- Dynamic instrumentation shows limited detail
- Limited set of compilers and libraries – problem for some large codes
- Modify build (make/cmake) scripts for best data collection

What's The Point?

- Light weight instrumentation and intelligent data reductions are **critical** to scaling up to exascale systems
- Easy to be swamped by the enormous volume of data generated by running large applications at scale
- However, current tools are hobbled by more mundane considerations
- Simplicity and robustness are far larger impediments to tool adoption
- Advanced features can be useful, but not if it makes the tool fragile, not at the expense of being flexible, robust, lightweight or scalable