What can/should we measure with benchmarks?

Jun Makino
Department of Planetology, Kobe University
FS2020 Project, RIKEN-CCS

SC18 BoF 107 Pros and Cons of HPCx benchmarks Nov 13

Overview

- **Last 40 years of HPC benchmarks**
- **Why HPL survived? (Or why everybody else has not?)**
- **Is HPL good enough? (Clearly not...)**
- **So what should be measured and how?**

Last 40 years of HPC benchmarks

- 1979: Original LINPACK benchmark
- 1989 PERFECT Club
- 1990 SLALOM
- 1991 HPL (Parallel LINPACK)
- 1991 NAS Parallel Benchmark
- 1993 Top 500
- 1996 SPEC HPC
- 2005 HPCC
- 2013 HPCG
- 2014 HPGMG

1991: NAS Parallel Benchmark

Quote from Baily (2005?):

> At the time (1991), the only widely available high-end benchmark was the scalable Linpack benchmark, which while it was useful and remains useful to this day, was not considered typical of most applications on NASA's supercomputers or at most other sites. One possible solution was to employ an actual large-scale application code, such as one of those being used by scientists using supercomputers at the NAS center.

In 2018, we are still talking about limitations of HPL and its possible replacement. Why?

Why HPL survived? (Or why everybody else has not?)

In my opinion, HPL actually has unique features not shared by other benchmarks

- **Only the problem to be solved is specified. Neither the source code nor even algorithms (Gustavson's recursive blocking was invented in 1997, after the start of Top 500)**
- **Problem size is not specified. Machines from kflops to yottaflops can be measured.**
- **One condition: the FP operation count for a given problem size must be fixed (no Strassen-Winograd or any other algorithms)**

Why HPL survived? (continued)

- Requirement for communication is modest and can be reduced by increasing the main meory size
- Requirement for main memory BF is also can be reduced by increasing the cache size
- Only the DGEMM kernel should be optimized (at least on usual multicore machines... For accelerators with extreme performance ratios, many other things matter — see JM et al. Procedia Computer Science 4 2011 888-897)

In short, HPL has survived because it can adopt its requirement (and algorithms and source code) to the change of architectures over several decades

Then is HPL good enough?

Clearly not — that's why there have been so many proposals for alternatives.

But what can we do? We now know

- **HPL does not represent application performance**
- **Fixed set of applications (or mini-applications) do not survive (PERFECT, SPEC HPC, ...)**
- **Fixed set of codes for basic algorithms do not survive (NPB, HPCC, ...)**

So what can we do?

What we are doing now:

a) **Define "representative" applications to measure the performance of a machine (in particular new one to be built)**

b) **Use HPL — certainly "not enough"**

c) **Use HPL + HPCx**

d) **Something else**

Problems with "representative" applications

- We know they would not survive (PERFECT, SPEC HPC, ...)
- They tend to be negative against new architectures (or anything new)
  - They are optimized to yesterdey's architecture, not to tomorrow's architecture.
  - They do not support emerging programming model
  - They do not use the latest algorithms

Problems with HPL + HPCx

- HPL measures the peak floating-point performance (call this f) and whether or not the rest of the system fills the minimum requirement of HPL (tradeoffs between cache and memory BW and between memory size and network BW)
- HPx measures something else (call this x)
- From the hardware point of view, for most types of x, trying to make x/f large results in the decrease of energy efficiency.
- The "optimal" value of x/f varies not just for different problems, but for a single problem depending on algorithms and implementations, and many researchers are trying to reduce necessary values of x/f.

The efforts to keep the x/f number high has negative impact on energy efficiency and tends to discourage the effort to improve algorithms

(Of course, DP-only HPL has its problem)

Problems with HPL + HPCx (2)

- The actual performance bottleneck of real application codes can come from many different points (or combinations of them) in the case of today's complex processors with wide SIMD units and multiple levels of cache shared by many cores. HPL (and HPCx in general) are too simple. Examples include: the number of architectural and physical registers, latencies and bandwidths in all levels of on-chip storage, the performance of random or stride access at all levels, the amount of available OOO resources.
- Our experience on K and Post-K suggests that the efficiency of many "real" applications are limited by neither the peak floating-point performance(HPL) nor the main memory bandwidth(HPCG) but something else.

My points:

1. **Benchmarks (or a set of them) should show how the hardware should be designed, not just for a single machine at one time, but for a reasonably long period.**
2. **HPL shows how we can design a long-lived benchmark.**
3. **In principle, other HPCx benchmarks can be designed to achieve similar long lifetimes.**
4. **However, on modern multicore processors with deep memory hierarchy, the efficiencies of many applications are not limited by what are measured by HPCx benchmarks.**
5. **We probably need new ways to define "application benchmarks" to allow larger rooms for optimization (define problems, not source code or algorithms)**

Or are there better ways?