A Framework for Measuring Productivity in HPC

SC03 panel

Marc Snir
What is **Productivity**?
- Amount (or value) of output per unit of input (e.g., cars per work hours)

Why do we care?
- To make the right resource allocations & have the right policies

How do we measure input into supercomputing platform?
- Cost
  - need to allocate correctly shared costs (cost allocation problem – e.g., using Shapley value)

How do we measure supercomputer output?
- Posit existence of *Utility function* [von Neumann & Morgenstern]

Productivity of platform is not (not only) intrinsic; it is a function of applications run on platform, needs/preferences of user, environment (e.g., programmers’ skills), etc.
Simple example: one platform, one output

Utility

Cost

Time

Productivity = U/C

Cost

Time

maximum productivity

\[
\Psi = \max_T \frac{U(T)}{\text{Cost}(T)}
\]

\[
\Psi = \max_C \frac{U(T(C))}{C}
\]
How much should one spend on tuning code?

\[ T = T_{\text{code}} + T_{\text{exec}} \; ; \; \; C = C_{\text{code}} + C_{\text{exec}} \]

Development time (cost) and execution time (cost) are not independent; any theory of productivity in HPC must address the tradeoff.

"Isocost" curves
From Description to Prescription

- Need ability to predict time-to-solution
  - for given problem, on given system, for given cost
    \[ T(P, S, C) \]

- Predict execution time, for given problem, on given platform and given programming effort
  \[ T_{\text{exec}}(P, S, H) \]

- Compare execution time, for given problem and given programming effort, across two platforms
  \[ T_{\text{exec}}(P, S_1, H) : T_{\text{exec}}(P, S_2, H) \]

“predict”, or “compare” to be interpreted very loosely: statistically meaningful predictor
Grand Dream

- Set of parameters (metrics) that characterize system $M_1(S), ..., M_m(S)$
  - characterize development and execution environment
  - some may be subjective
- Set of parameters (metrics) that characterize application $N_1(P), ..., N_n(P)$
  - characterize coding and execution complexity
- Predictor

$$T_{exec}(P, S, H) = T(N_1(P), ..., N_n(P), M_1(S), ..., M_m(S), H)$$
Awakening

- Grand dream realizable to some extent, for environments where tuning is negligible fraction of code development effort
  - performance prediction
  - SE predictive models of coding effort
- We lack understanding (in SE sense) of HPC software development process
- We lack understanding of dependence between coding effort and performance (tuning)
Modest Experiments

- Compare $T_{\text{exec}}(P, S_1, H) : T_{\text{exec}}(P, S_2, H)$

where $S_1$ and $S_2$ differ in only one aspect (e.g., MPI vs. UPC) and $P$ is simple, well-defined problem

  - can look at other “measures of goodness”, in addition to execution time

- Develop stopping rules deciding when to stop tuning code in large projects.

  - analogous to rules applied to code debugging

- Develop time-limited benchmarking competitions?