Introduction to Parallel Processing

Lecture 10 : Performance Modeling

Professor Amanda Bienz

What is Profiling?

- Dynamic program analysis
- Measures various components of program, such as:
  - Memory usage
  - Cache hits / misses
  - Duration of function calls
- Useful in program optimization

Conjugate Gradient

$$
\begin{aligned}
&x_0 = \text{initial guess;} \\
&r_0 = b - Ax_0; \\
&p_0 = r_0; \\
&\textbf{for } i = 0, 1, 2, \ldots \textbf{ do } \{ \\
&\qquad \alpha_i = \frac{r_i^\mathsf{T} r_i}{p_i^\mathsf{T} A p_i}; \\
&\qquad x_{i+1} = x_i + \alpha_i p_i; \\
&\qquad r_{i+1} = r_i - \alpha_i A p_i; \\
&\qquad p_{i+1} = r_{i+1} + \frac{r_{i+1}^\mathsf{T} r_{i+1}}{r_i^\mathsf{T} r_i} p_i; \\
&\}
\end{aligned}
$$

SpMV

Two inner products

SpMV A*p_i

Conjugate Gradient Profiling

$$
\begin{aligned}
&x_0 = \text{initial guess};\\
&r_0 = b - \boxed{Ax_0;}\\
&p_0 = r_0;\\
&\textbf{for } i = 0, 1, 2, \ldots \textbf{ do } \{\\
&\qquad \alpha_i = \frac{\boxed{r_i^\top r_i}}{\boxed{p_i^\top \boxed{Ap_i}}};\\
&\qquad x_{i+1} = x_i + \alpha_i p_i;\\
&\qquad r_{i+1} = r_i - \alpha_i A p_i;\\
&\qquad p_{i+1} = r_{i+1} + \frac{r_{i+1}^\top r_{i+1}}{r_i^\top r_i} p_i;\\
&\}
\end{aligned}
$$

Profiling with Timers:

- Laplacian (simple) matrix on 16 processes
- Measure SpMVs to require 3.2e-01 seconds
- Measure Inner Products to require 1.2e00 seconds

- Wouldn’t expect inner products to take longer than SpMV
- And, in fact, they probably don’t

Valgrind

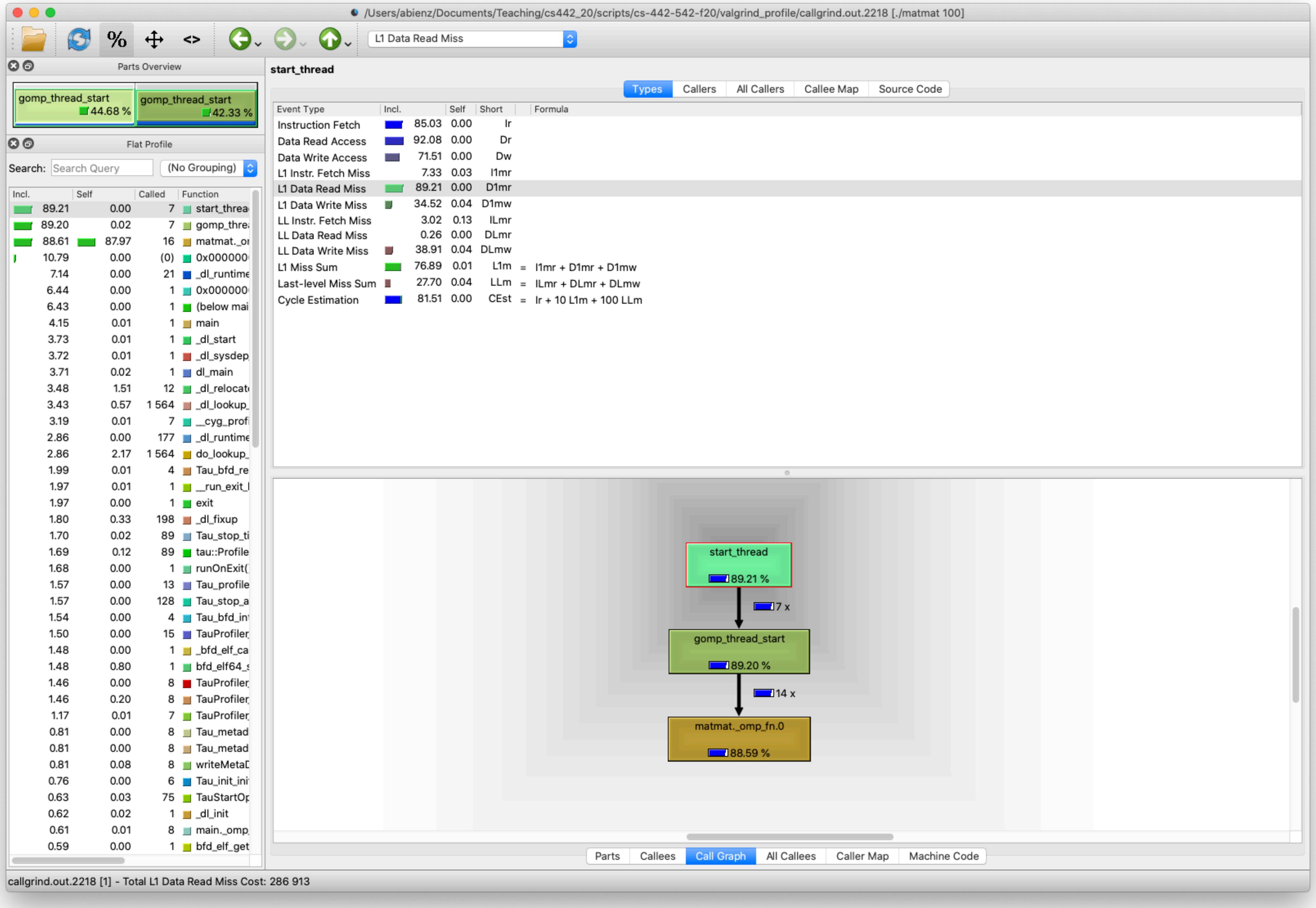
- If you use a programming language without garbage cleanup, I **highly** recommend checking out Valgrind : https://www.valgrind.org
- Debugs and profiles
  - Detects memory management bugs
    - Memory leaks (forget to free data)
    - Accessing unallocated data
  - Detailed profiling can help find program bottlenecks
- Simple command 'valgrind <valgrind_args> ./<filename> <args>'

Cachegrind

- Part of Valgrind
- Measures cache hits / cache misses and cache hit rates
- This can be very helpful in determining if your program is utilizing cache
- ‘valgrind —tool=cachegrind ./<filename> <fileargs>

QCachegrind Profiler

- ‘valgrind —tool=callgrind —dump_instr=yes —simulate-cache=yes —collect-jumps=yes ./<filename> <fileargs>



Tau Profiler

- Easy to install (I was able to download and use it on Wheeler)
  https://www.cs.uoregon.edu/research/tau/home.php

- Profiles how much time is spent in different functions, less about cache hits / misses

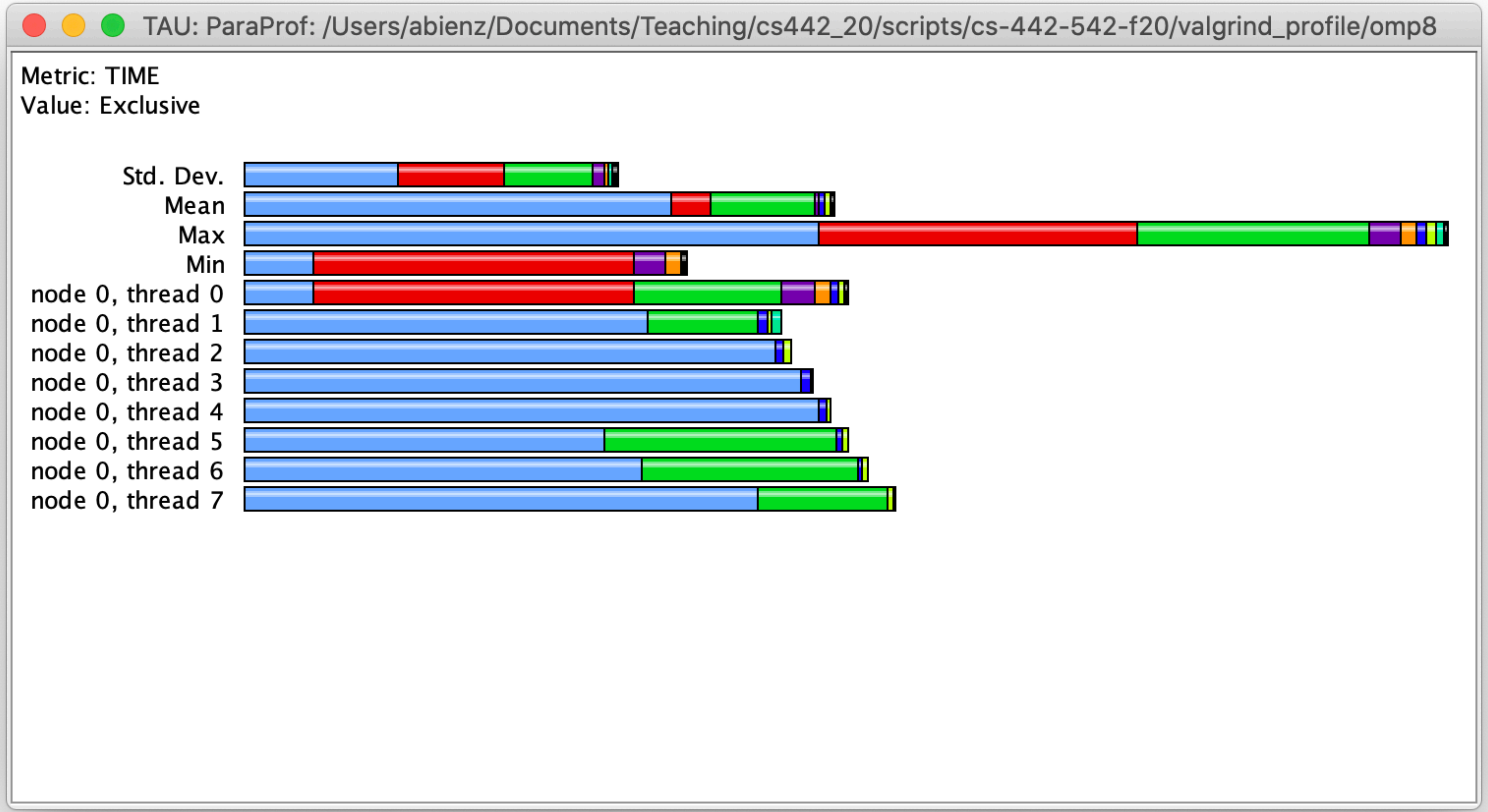
- Need to export TAU_MAKEFILE to appropriate file, for instance the version of Tau that is installed with MPI support

- Then run program with tau_exec to get profiles:
  mpirun -n <np> tau_exec ./filename <file_args>

- Can run MPI code with various process counts to get a variety of profiles

Paraprof

- Comes with TAU
- Visualization tool for profiles



Tracing

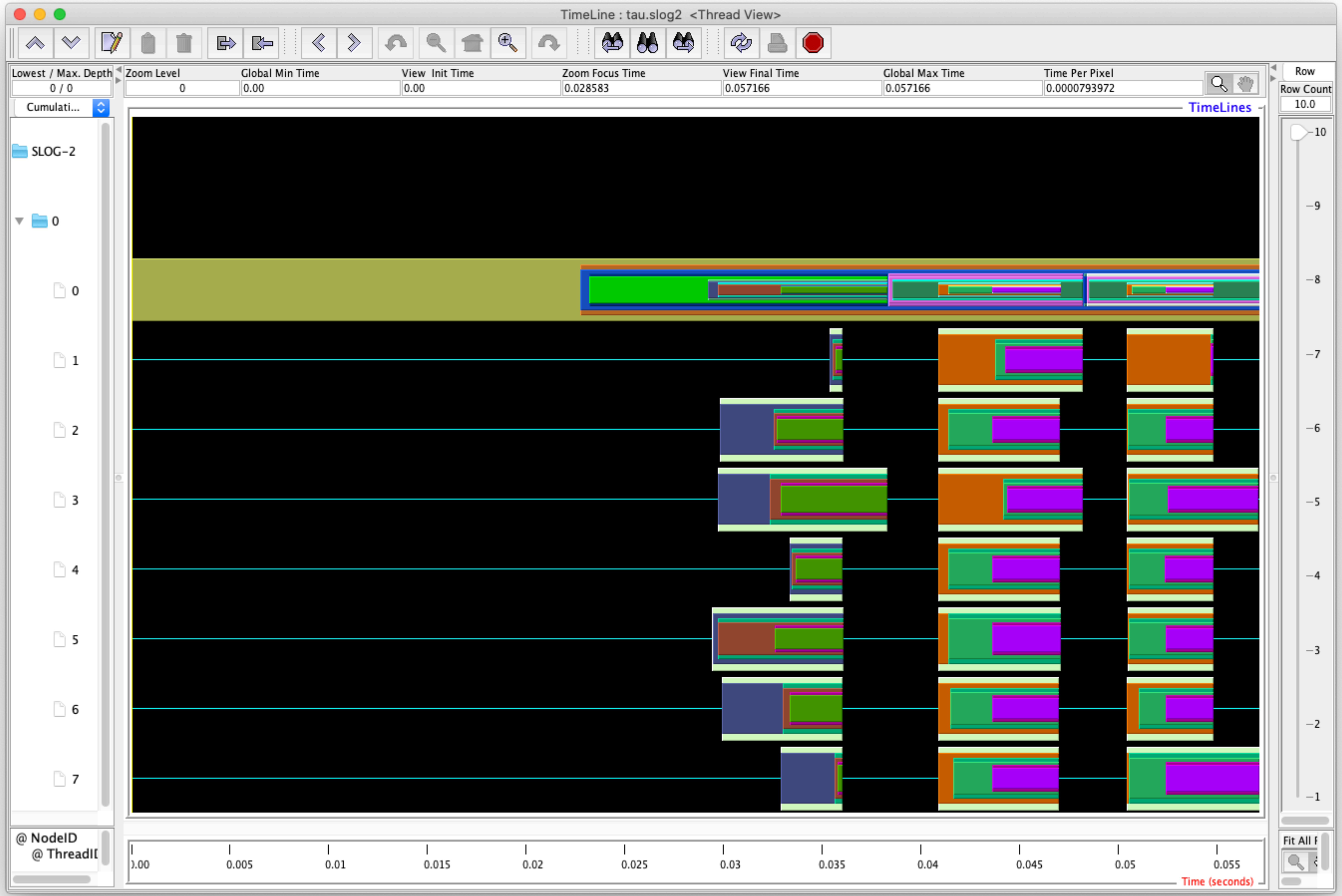
- Shows when are where performance is achieved
- Often can be shown as timeline (what happened at each instance during your program)
- Rather than having functions display by duration, displays functions as they are executed during timeline
- In my opinion, this is often most useful way to find performance issues with code, particularly in parallel

Tracing with TAU

- Compile program with tau_cc.sh as before
- Before running program export TAU_TRACE=1
- Get profiles (as normal), an events.0.edf, and files called tautrace
- To visualize:
    - tau_treemerge.pl
    - tau2slog2 tau.trc tau.edf -o tau.slog2
    - jumpshot tau.slog2

Jumpshot

- Comes with TAU to visualize tracing



Downside of Profilers (Tracing)

- Need to trace each program (every different matrix running on different numbers of processes) to accurately analyze bottlenecks
- Time consuming (and uses a lot of compute power)
- Idea: Would like to be able to model the performance of our application
  - Analyze how we expect the application to perform based on small benchmarks

Typically, Data Movement is Most Expensive

- For sparse methods, like CG, we can assume majority of cost is in data movement
- So, measure cost of communication in:
  - 1. SpMV
  - 2. Inner Project
- First, how to measure these costs, then we will look at how to extrapolate this measurement to any matrix / number of processes

Back to Performance Modeling

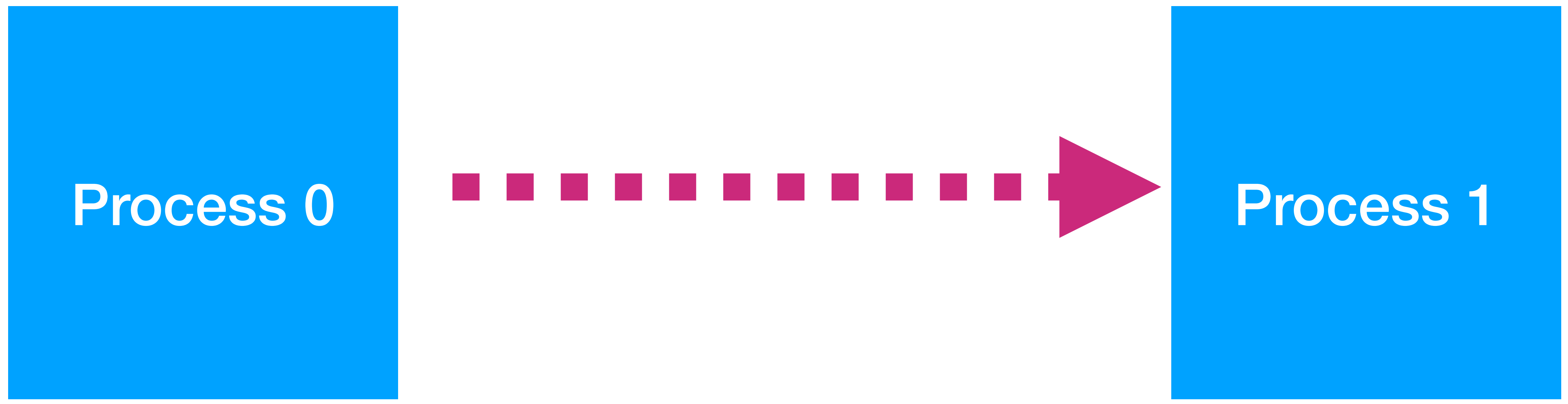
- Use performance models to analyze performance of components of program
- $T = \boxed{\alpha \cdot n + \beta \cdot s} + \boxed{\omega \cdot c}$
- Postal model for communication plus simple model for computation

Model Parameters

- How do we get parameters for model?
  - $T = \boxed{\alpha \cdot n + \beta \cdot s} + \boxed{\omega \cdot c}$
  - Computation : STREAM or similar memory benchmark
  - Communication : Can use OSU benchmarks, but usually easiest to measure yourself

Communication Model Parameters

- How do we measure cost of a single message?



1. Start Timer
2. Send Message

3. Receive Message
4. End Timer

- What is the problem here?

Communication Model Parameters

- Solution : Ping-Pong Tests (like your homework)

1. Start Timer
2. Send Message
3. Receive Message
4. End Timer

- **Must wait for first message to be received before second is sent!**

Communication Model Parameters

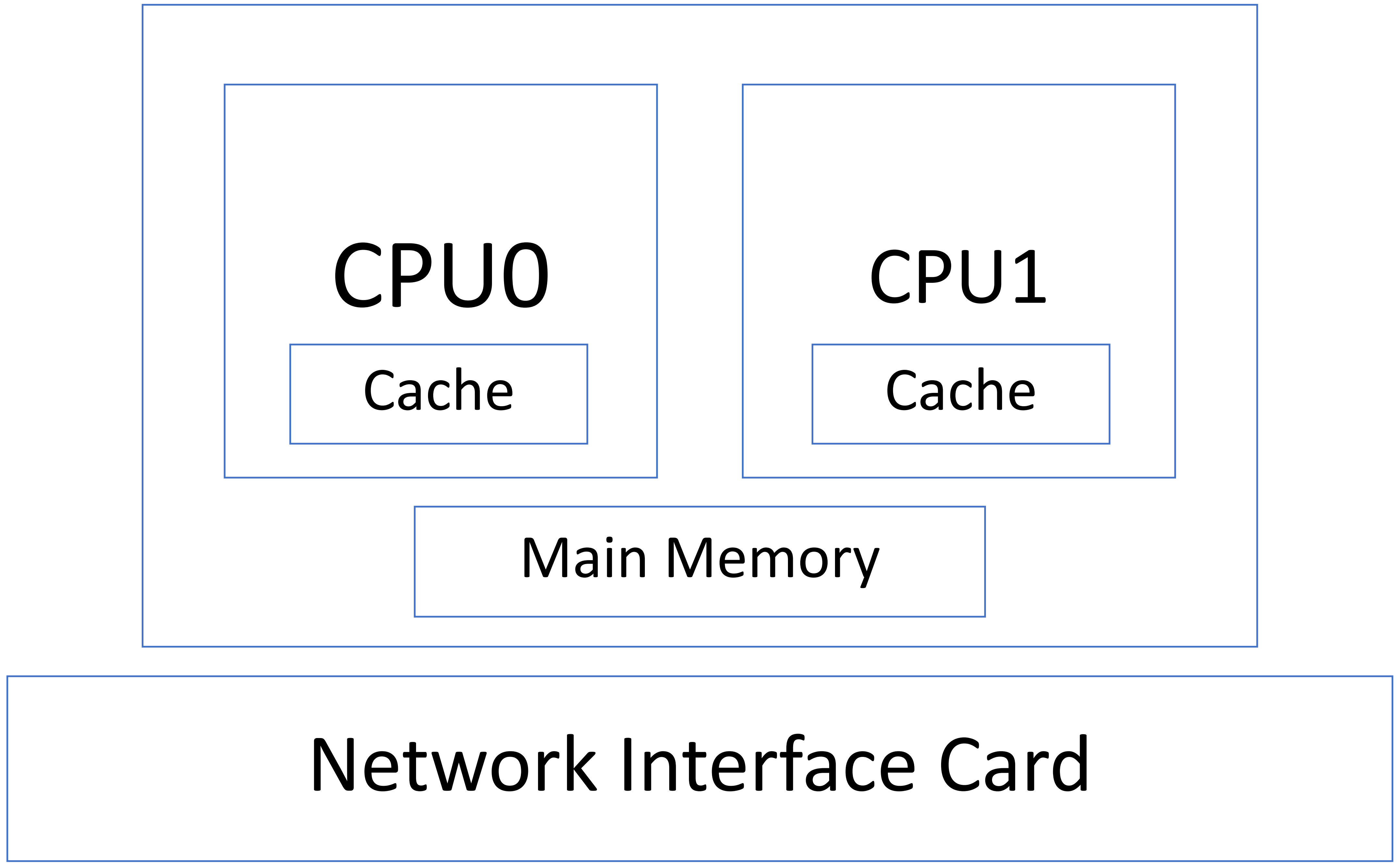
- How do we actually get alpha and beta from these timings?
  - Can calculate them in Python (or language of your choice)
  - Solve a simple linear system
    - Have a bunch of timings
    - Each send one message
    - Each message size is different
    - Add all of these to a matrix and solve system

$$\begin{bmatrix} 1 & t1 \\ 1 & t2 \\ 1 & t3 \\ & \ldots \\ 1 & tn \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

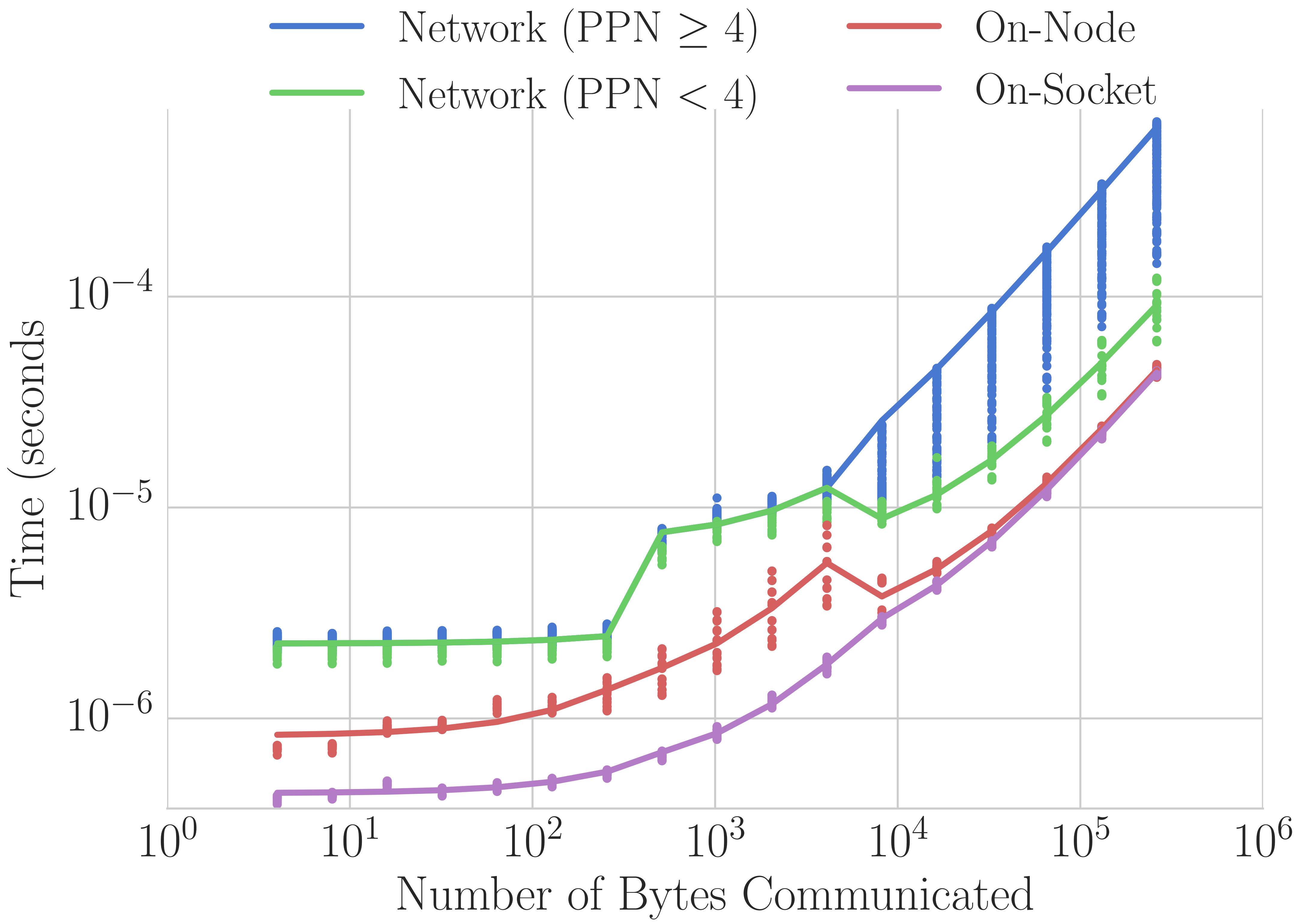
Performance Model Improvements

- Max-Rate Model : include injection bandwidth
- $$T = \alpha \cdot n + \frac{\mathrm{ppn} \cdot s}{\min(R_N, R_p \cdot \mathrm{ppn})}$$
  - ppn : number of processes per node
  - R_p : inter-process bandwidth
  - R_N : injection bandwidth

Performance Model Improvements



Symmetric Multiprocessing (SMP) Node

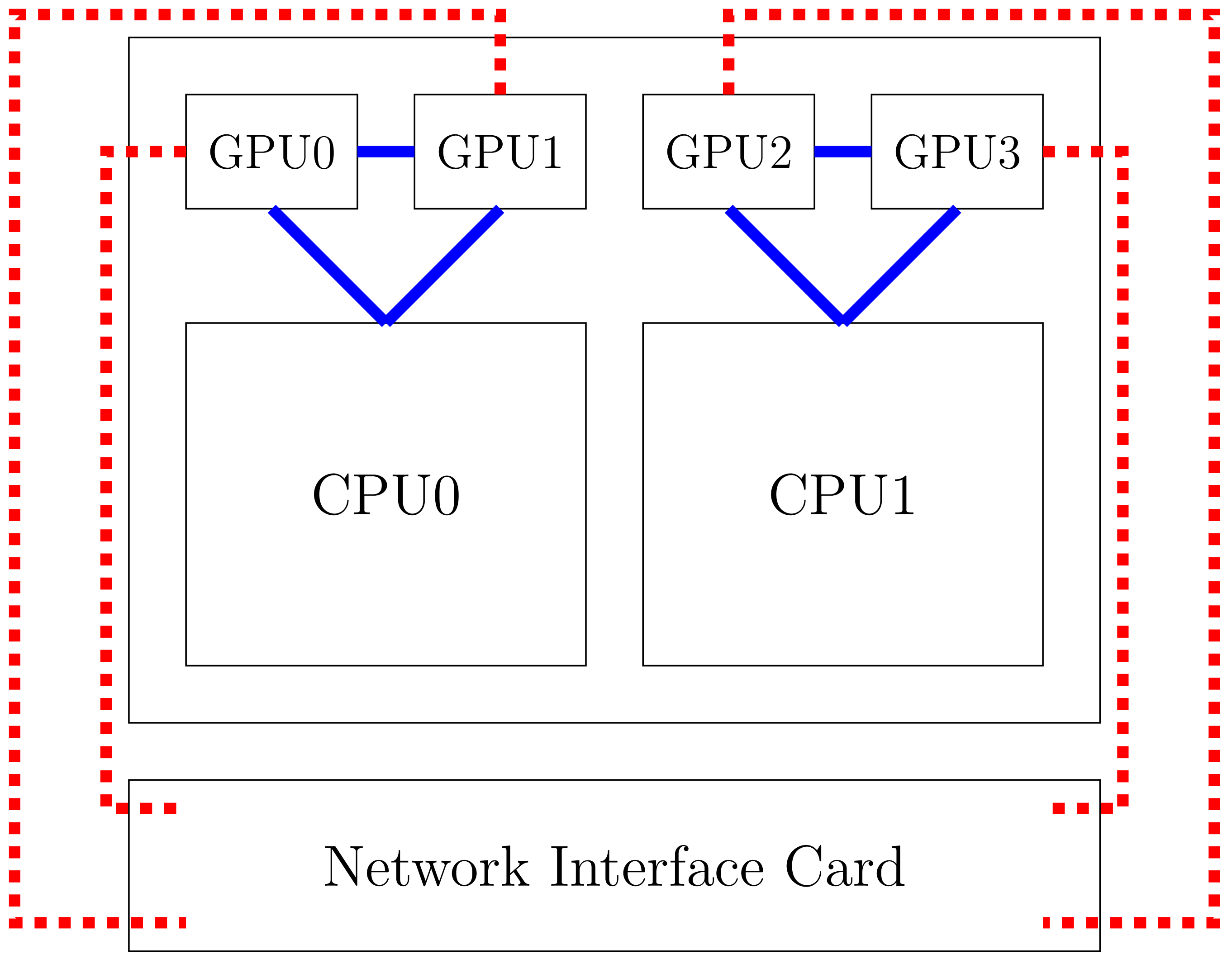


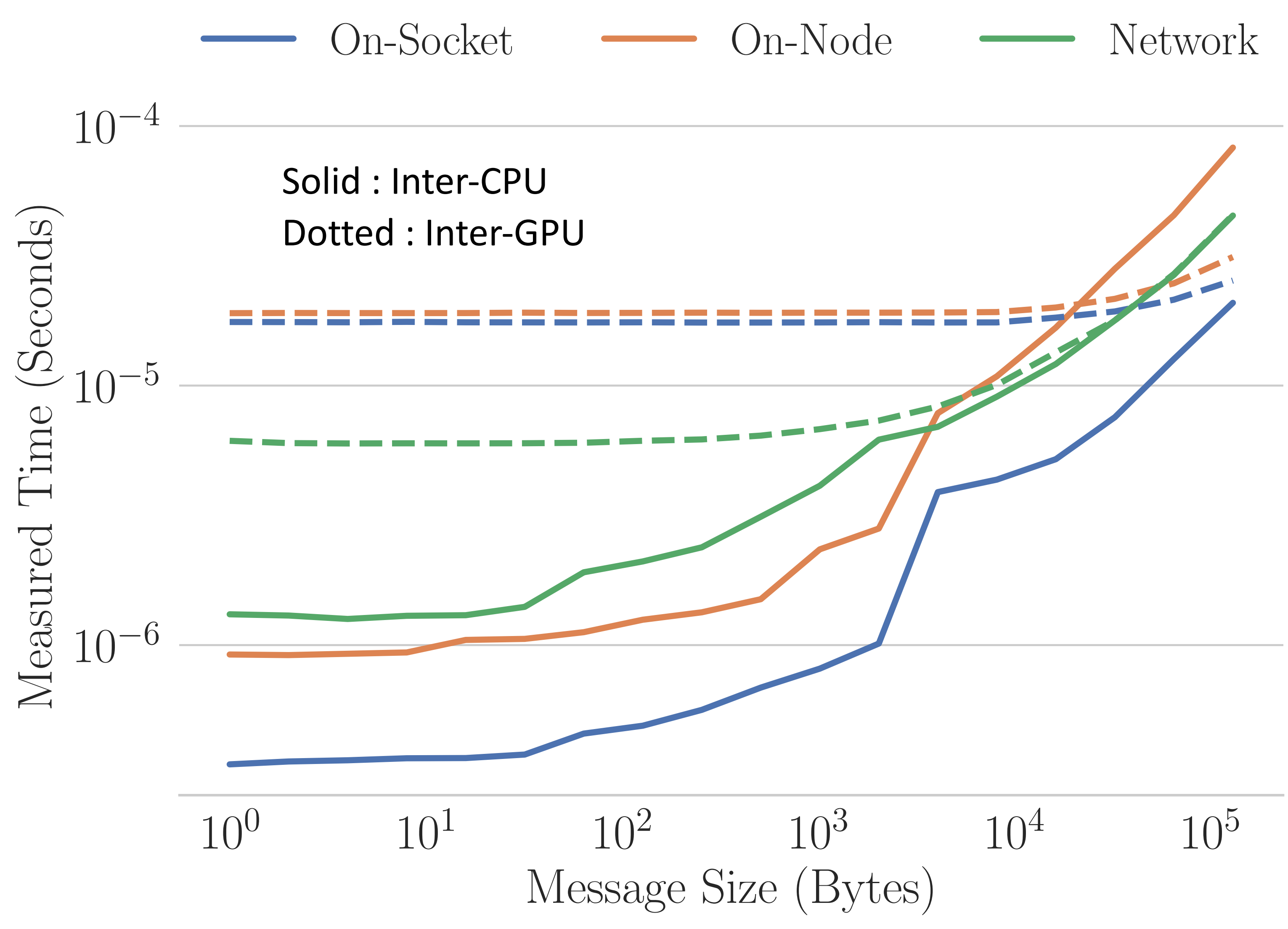
Additional Model Improvements

- Irregular communication : large number of (sometimes large) messages
  - Queue search (matching) costs
  - Network contention costs
- Collective algorithms : synchronization costs
  - LogP Model
  - LogGP Model

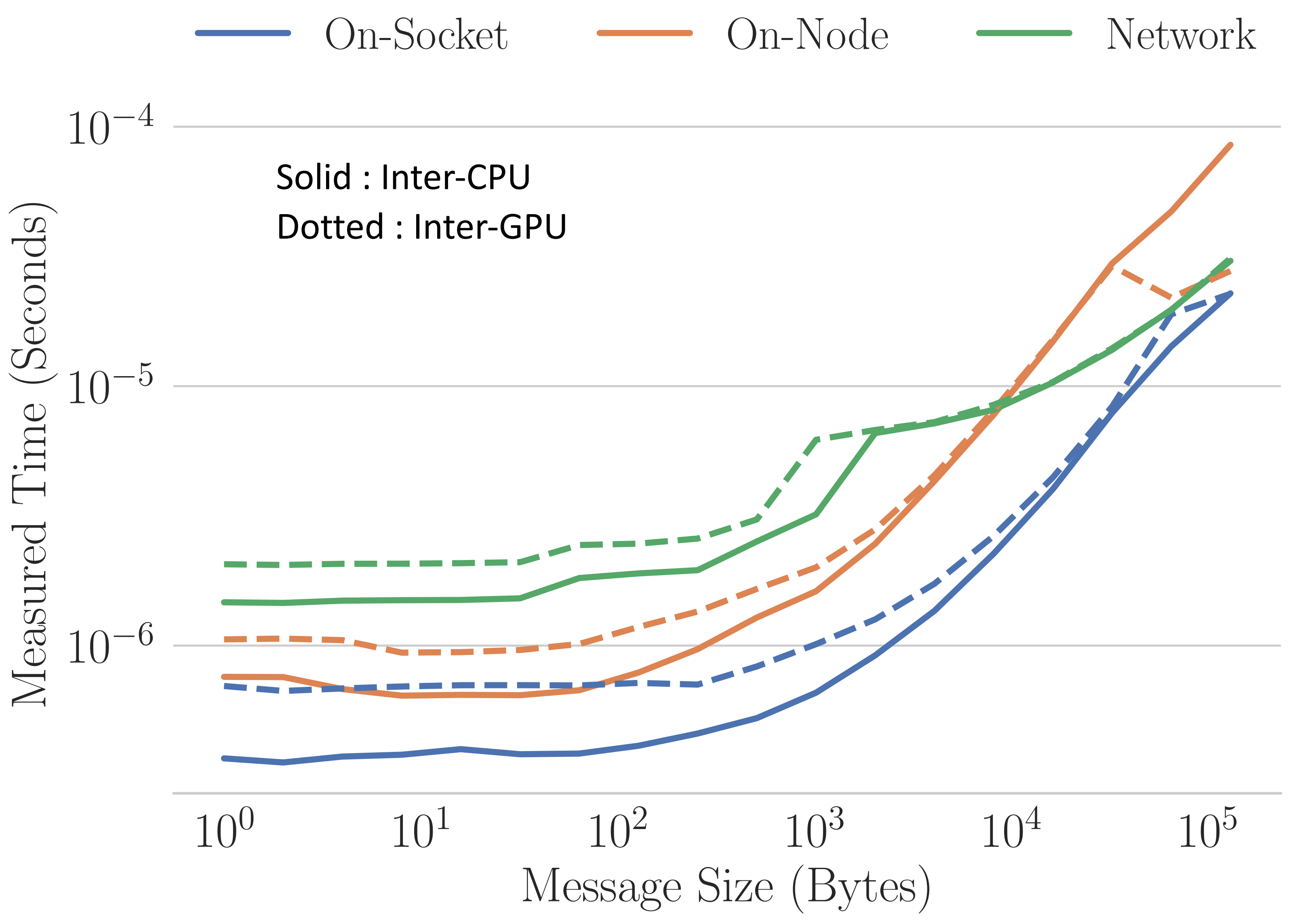
Importance of Models

- Lassen (Sierra) and Summit : two Top500 supercomputers
    - Heterogeneous (GPUs on node)





Lassen, SpectrumMPI



Lassen, MVAPICH2-GDR