Title:	Matriarch: A Framework for Automated Performance Analysis
Author(s):	Marcus, Ryan C.
Intended for:	LANL CS/IT Mini-showcase, 2014-07-31 (Los Alamos, New Mexico, United States)
Issued:	2014-07-24

2014 Computing and Information Technology Student Mini Showcase Abstract

(Due July 25)

Student(s) Information

Name	University/College	Email Address
Ryan Marcus	University of Arizona	rmarcus@lanl.gov

Mentor(s) Information

Name	Organization	Email Address
Cornell Wright	**HPC-5**	cornell@lanl.gov

Select one or both: **Poster** ☒ **Technical Talk** ☐

Topical Computing Area *(i.e. Data Intensive Computing, Networking, Computer Security, File Systems, etc.)*
High Performance Computing

LA-UR Number:

Title
Matriarch: A Framework for Automated Performance Analysis

Abstract

The performance of a particular HPC code depends on a multitude of variables, including compiler selection, optimization flags, OpenMP pool size, file system load, memory usage, MPI configuration, etc. As a result of this complexity, current predictive models have limited applicability, especially at scale.

We present Matriarch, a tool designed to aid developers in building accurate predictive models of complex HPC codes. Matriarch provides a framework for repeatedly compiling, scheduling, monitoring, and analyzing codes on modern HPC systems like those found at Los Alamos. We have used Matriarch to quickly perform weak scaling studies, strong scaling studies, hotspot profiling, MPI tracing, and performance regression analysis on LANL's xRage, a hydrodynamics code.

As a result, we have identified scalability issues within hotspots, located MPI communication imbalances (due to AMR and domain decomposition biasing), and developed models that accurately predict xRage's performance at scale.

These insights provide valuable guidance for software developers seeking to optimize their codes, as well as for HPC engineers seeking to better understand a system's bottlenecks and resource limitations.

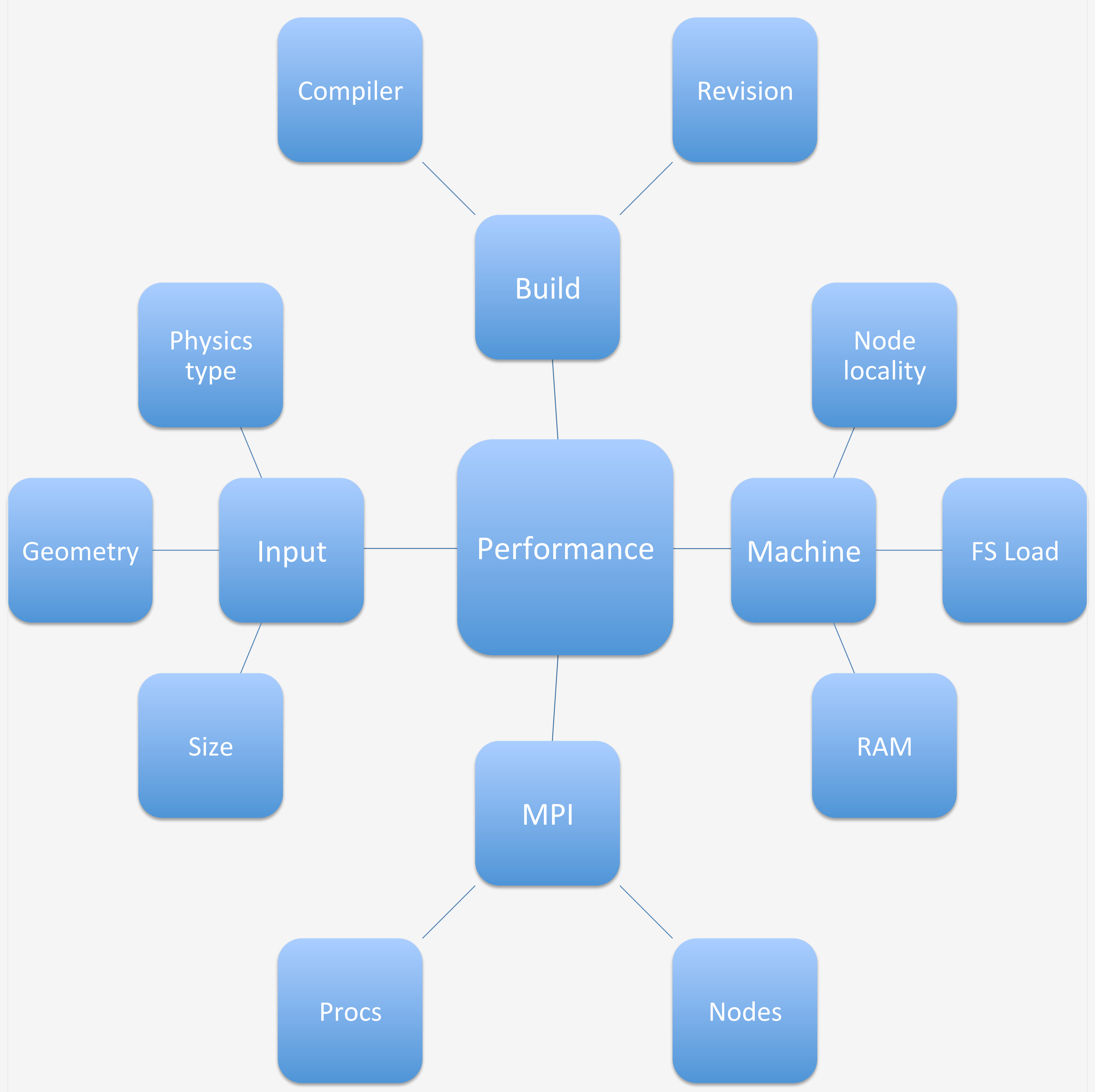
Return to mini-showcase@lanl.gov

Variables Affecting Code Performance

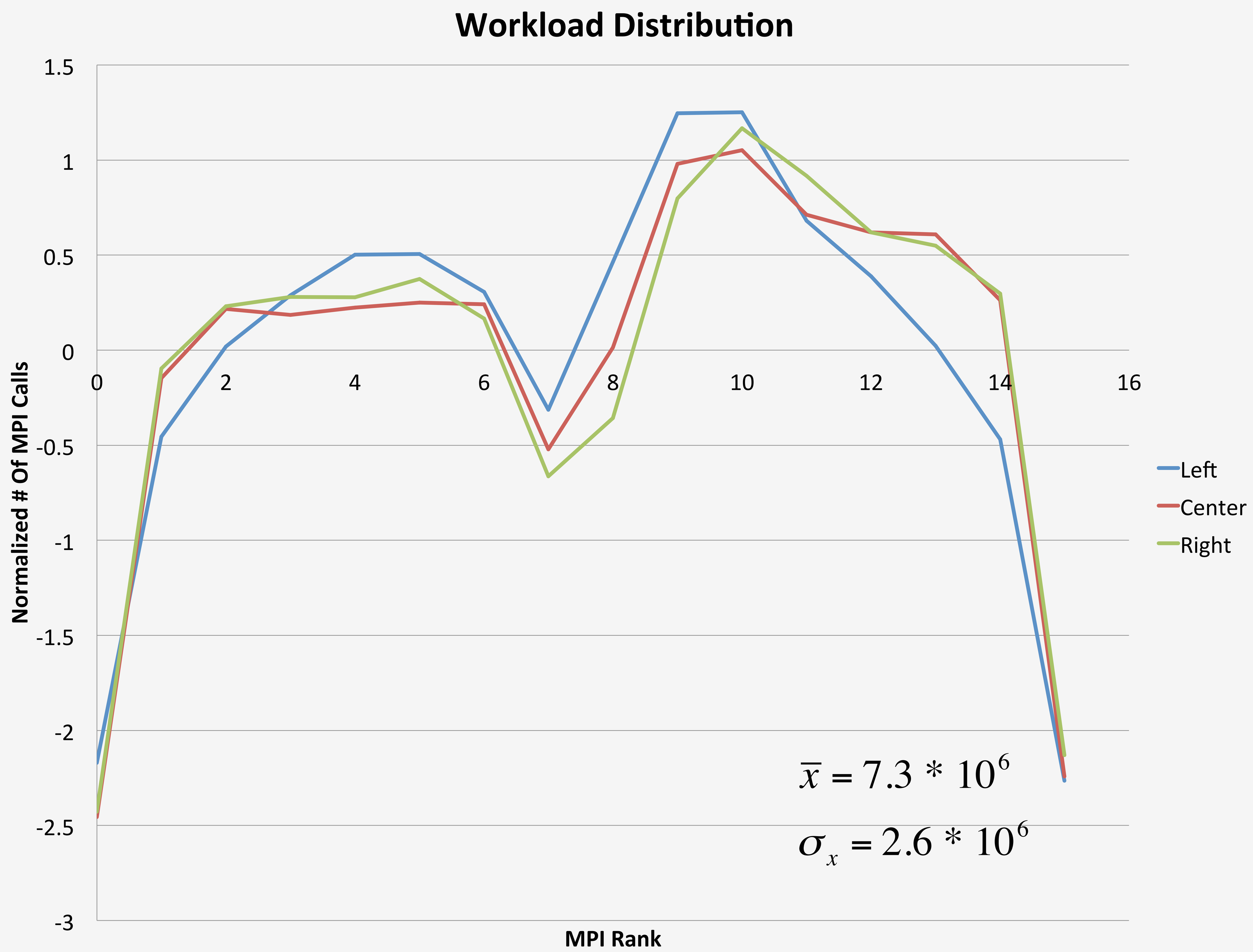
Matriarch is a framework for analyzing the behavior of high performance computing codes at every scale.

Increasing complexity in HPC has resulted in increased code complexity, making optimization a much more difficult problem. Developing a thorough understanding of scientific codes enhances the laboratory's ability to meet mission-critical goals while saving valuable energy.

Matriarch provides time-saving tools and automated workflows that work well with LANL's HPC systems.



Measuring Workload Distribution



Evenly distributing uneven workloads across many compute nodes is a difficult task. Suboptimal domain decomposition can slow down your code.

Often, the geometry and layout of a particular problem creates a higher workload in some areas of the domain than others. Optimally distributing an application's workload can be an incredibly complicated task. The above graph shows the normalized number of MPI calls made by each rank of the xRage hydrodynamics code running a cylindrical Sedov problem. The cylinder is shown centered, shifted to the left, and shifted to right to demonstrate how changes in problem geometry can affect workload distribution.

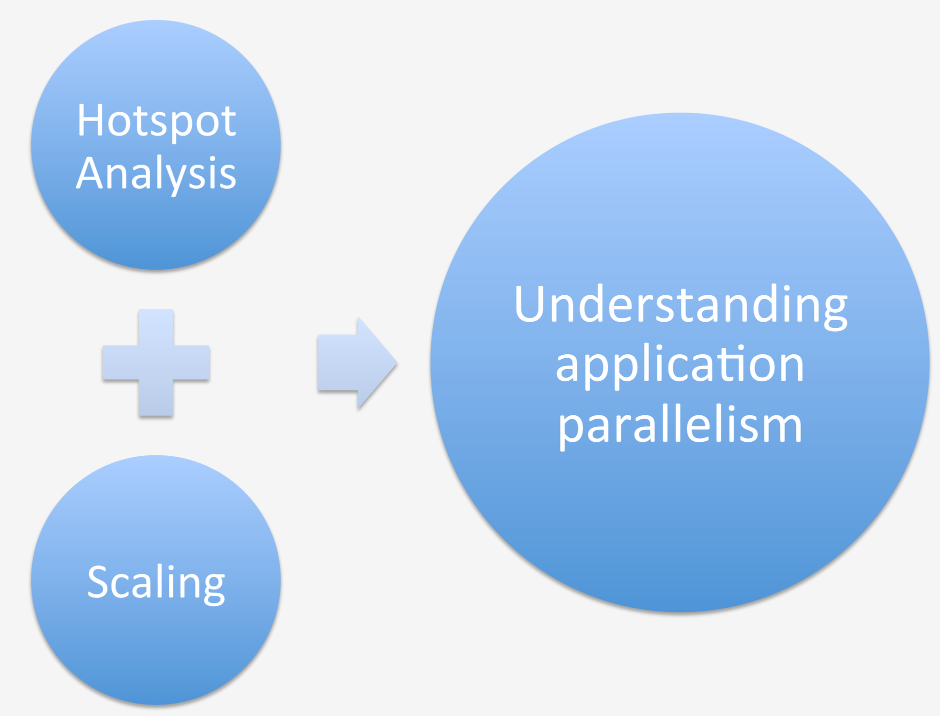
Hotspot Analysis

vtune1 | vtuneB | vtune_large_1

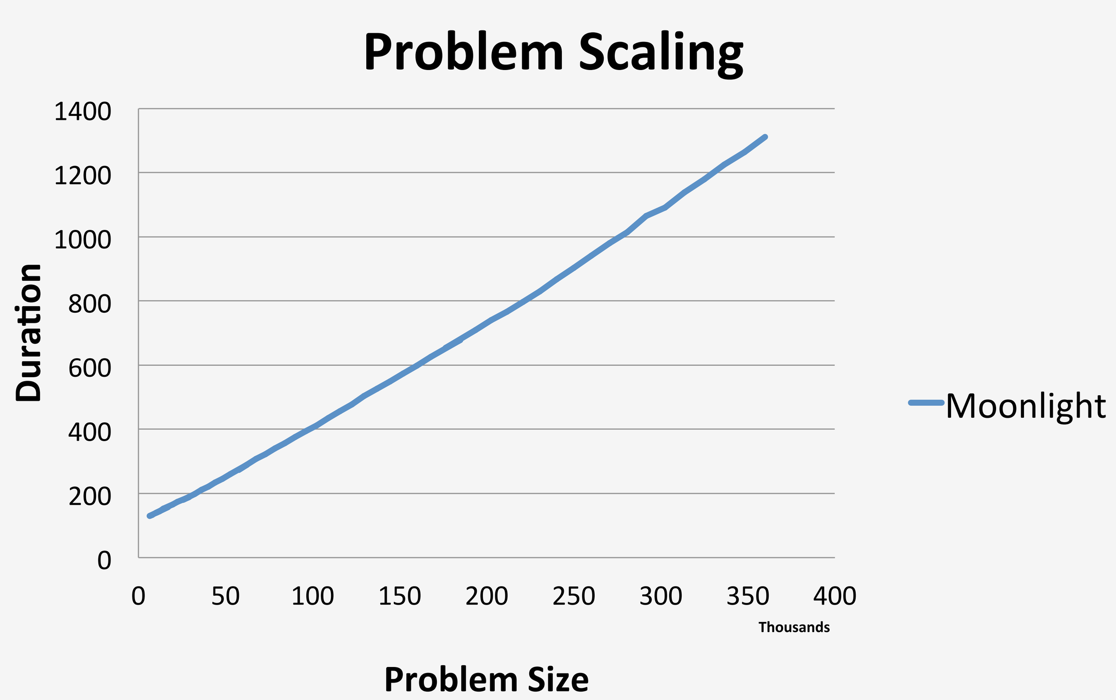
Function	Time	Function	Time	Function	Time
inside_com3	13.145760	– inside_com3	19.855494	– inside_com3	17.332948
ips_ptl_poll	9.610658	↑ mpi_waitall_f	11.867659	– mpi_waitall_f	12.833236
psmi_cycles_left	8.174111	↓ ips_ptl_poll	10.651356	↑ psm_mq_ipeek	12.220674
mpi_waitall_f	8.172550	↑ psm_mq_ipeek	10.500921	↓ ips_ptl_poll	11.985746
psm_mq_ipeek	4.968016	↑ whichcells_nu	5.271141	– whichcells_nu	4.779956
inside_com1	3.862118	– inside_com1	4.416622	– inside_com1	4.143702
whichcells_nu	3.847121	↑ inside_com2	3.666425	– inside_com2	3.508700
xeosg	3.202133	↑ hydro_mhd_riemn	3.033592	– hydro_mhd_riemn	2.719899
inside_com2	2.884646	↑ func@0xd840	2.315631	↑ MPI_ALLREDUCE	2.541764
mpi_wtime_	2.550854	↑ psmi_poll_internal	2.175335	↓ func@0xd840	2.518343

Matriarch enables a developer to compare hotspots as the problem scales.

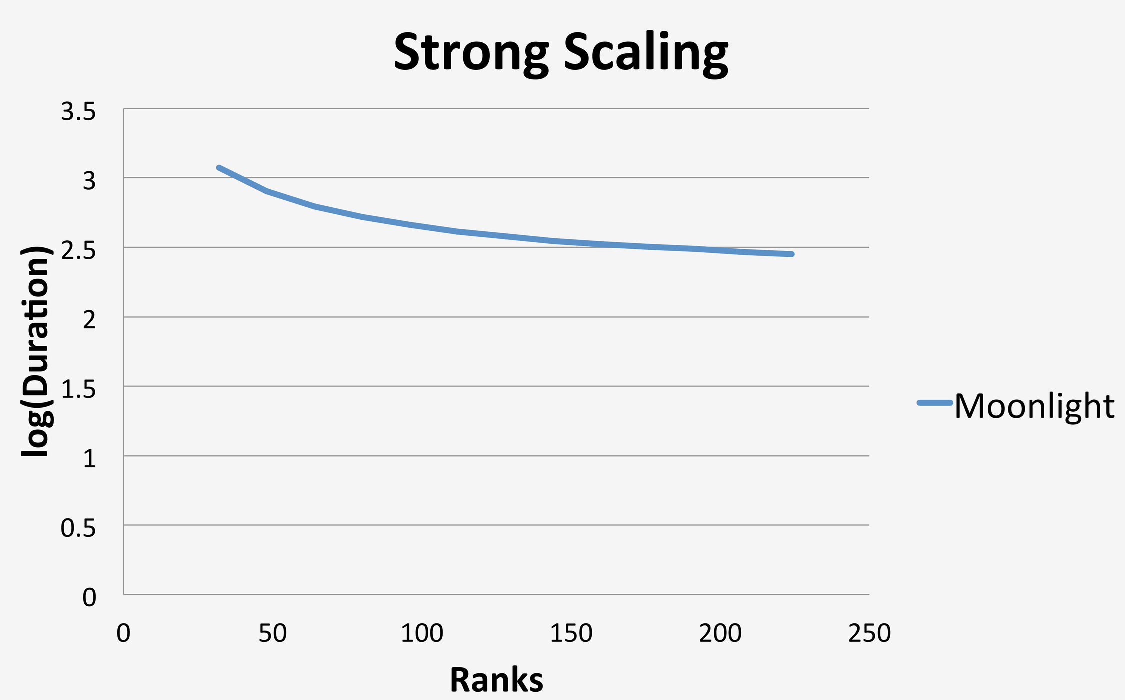
Traditional hotspot analysis can be useful, but looking at how the hotspots of a code change based on problem size or scale can determine how parts of a code scale. Here, we show hotspots in xRage as problem size increases. Rows highlighted in green represent an improvement, while rows highlighted in red represent a degradation.



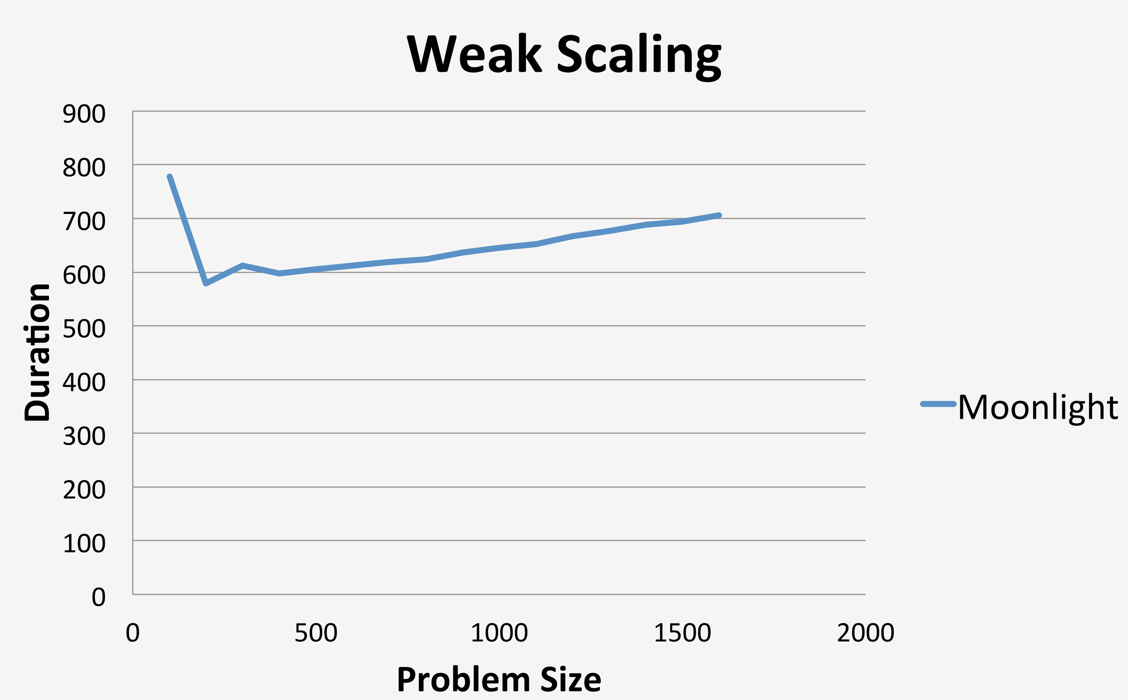
Problem Scaling, Strong Scaling, Weak Scaling, and Regression Analysis



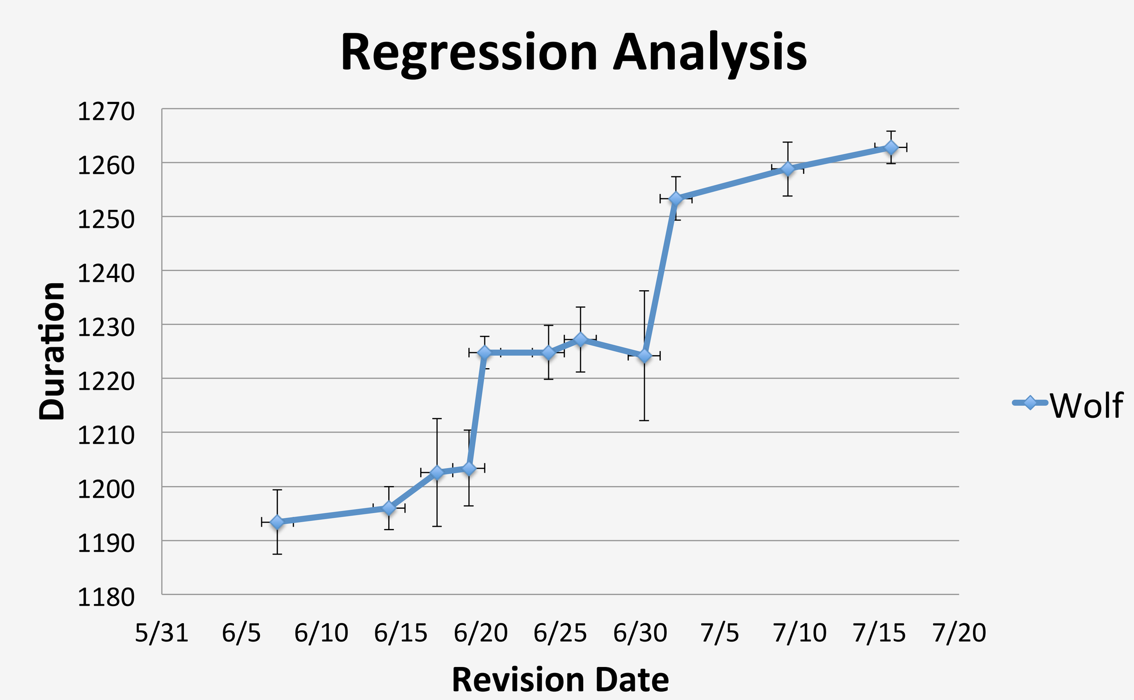
How does problem size affect duration, keeping the number of nodes the same?



How does the number of processors affect duration, keeping the problem size the same?



How does increasing the number of processors and the problem size proportionally affect duration?



How has the performance of the code changed over time?

Los Alamos NATIONAL LABORATORY EST. 1943