SC22

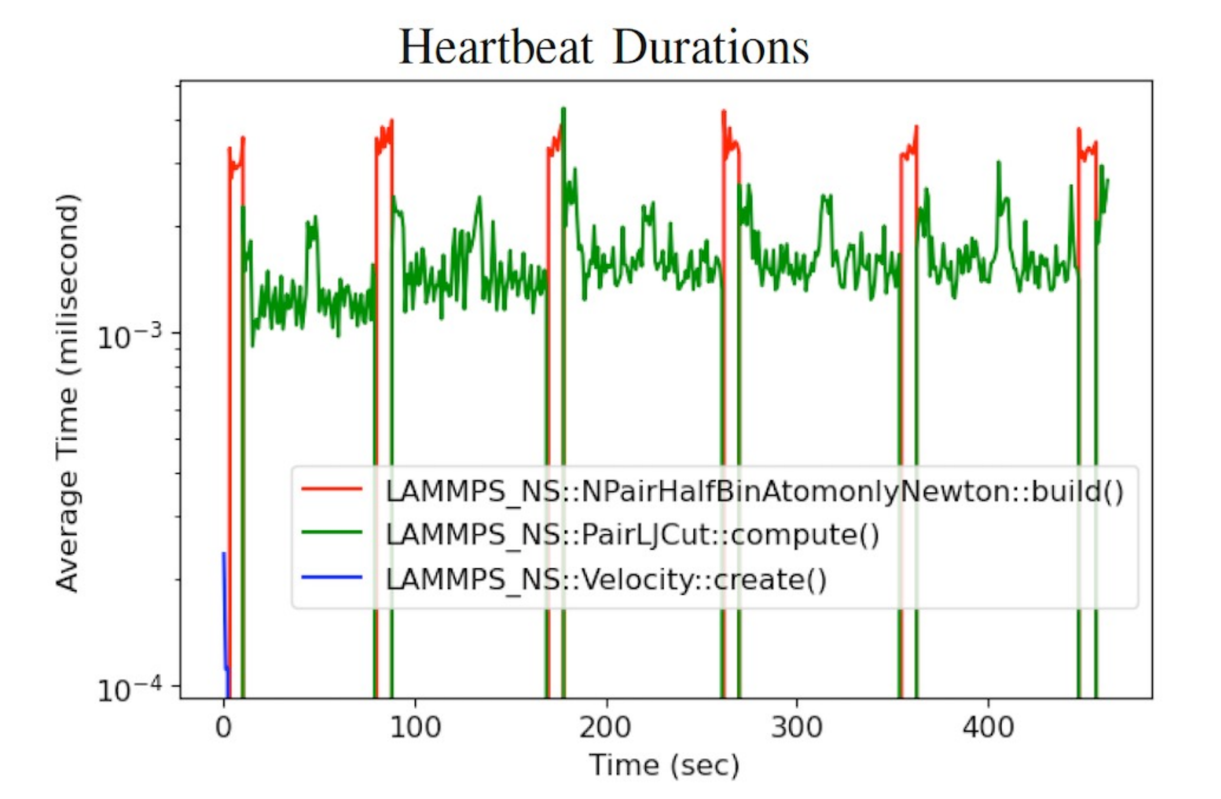
Dallas, TX | hpc accelerates.

AppEKG: A Simple Unifying View of HPC Applications in Production

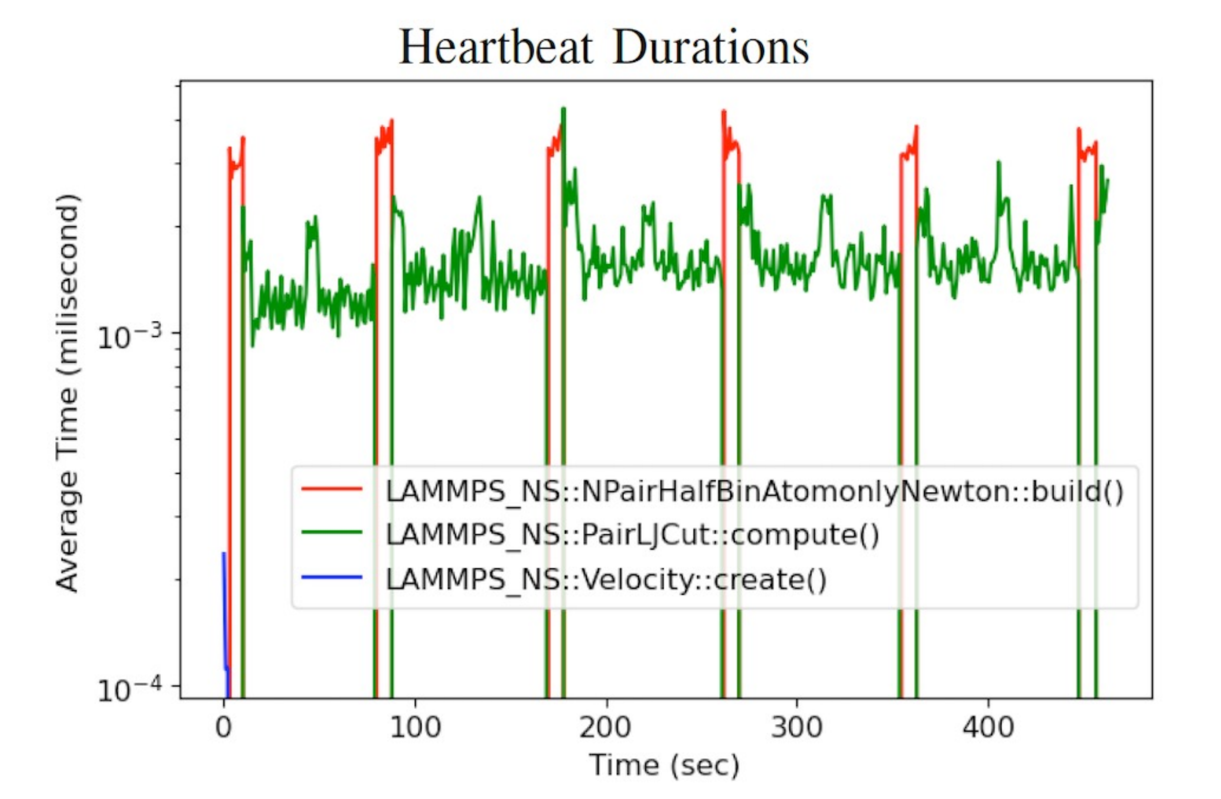
Authors

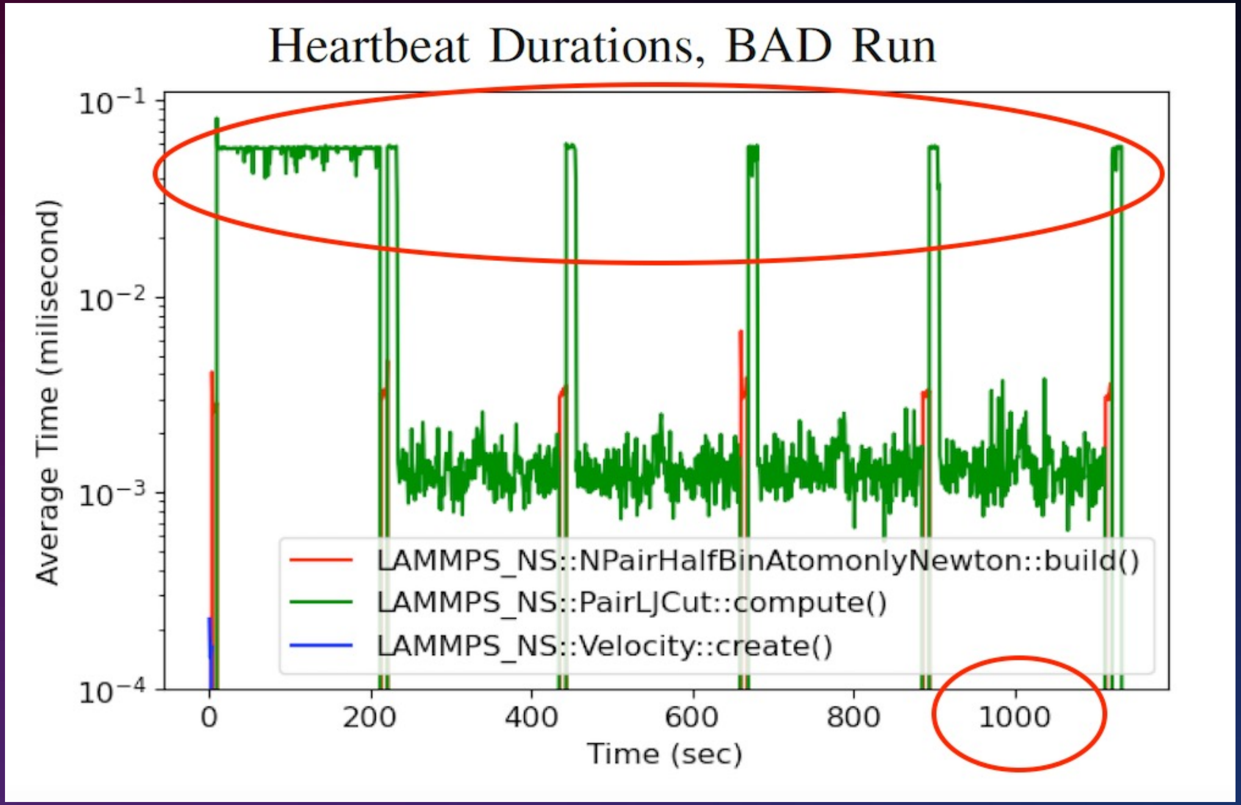
Mohammad Al-Tahat Strahinja Trecakov Jonathan Cook

LAMMPS Heartbeats Data



LAMMPS Heartbeats Data





AppEKG: A Heartbeat Framework

- The main goal of this research is to observe the application performance **in production** at the phase level.
- AppEKG collects heartbeat data (counts and average durations) from the representative phases.
- AppEKG writes out heartbeat data per process (rank) and per thread.
- Collected heartbeat data can be used to analyze and understand application performance in production.

APPEKG Macro Interface

EKG_BEGIN_HEARTBEAT(id, rateFactor)

EKG_END_HEARTBEAT(id)

EKG_PULSE_HEARTBEAT(id, rateFactor)

EKG_INITIALIZE(numHeartbeats, samplingInterval, appid, jobid, rank, silent)

EKG_FINALIZE()

EKG_DISABLE()

EKG_ENABLE()

EKG_NAME_HEARTBEAT(id, name)

EKG_IDOF_HEARTBEAT(name)

EKG_NAMEOF_HEARTBEAT(id)

Controlling Overhead: Sampling Interval

- AppeKG must limit I/O in order to control overhead.
- AppEKG accumulates heartbeat data internally over a pre-defined interval.
- Only writes out heartbeat counts and average durations per interval.
- The sampled data still captures the dynamic behavior of the applications.

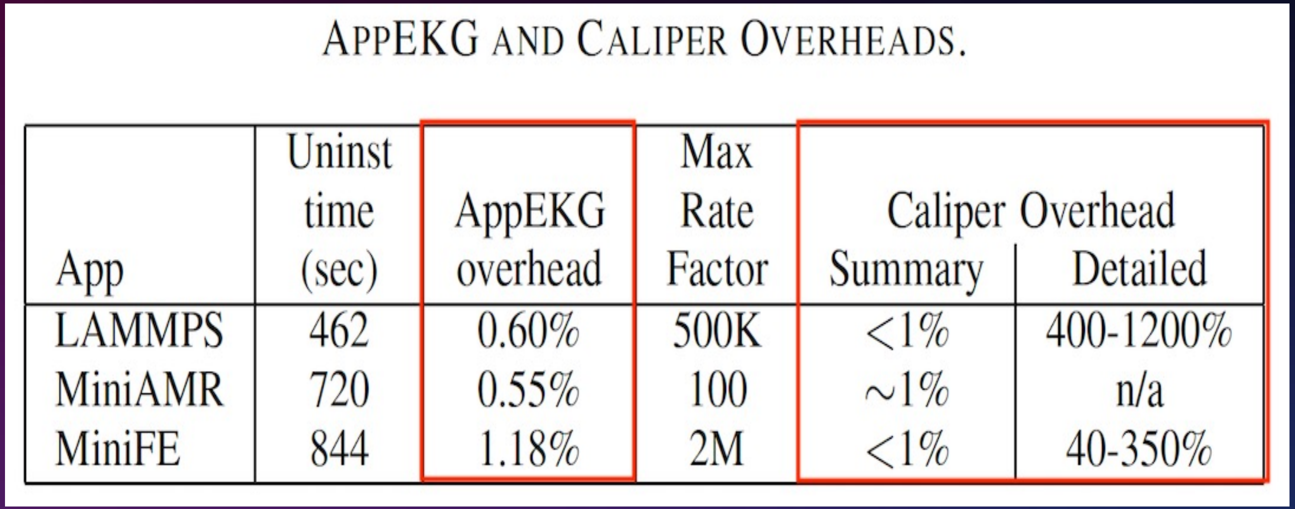
Controlling Overhead: Rate Factor

- Instrumenting a piece of code that has a high execution rate may produce high overhead.
- To control such overhead, a per-heartbeat rate factor is used to limit how often a heartbeat is produced.
- Implemented in the macro interface to avoid function call overhead.

```
#define EKG_BEGIN_HEARTBEAT(id, rateFactor)
…
   if ((_ekgHBCount[tid]++) % (rateFactor) == 0) {
                  _ekgBeginHeartbeat((id));
…
```

AppEKG vs Caliper

- AppEKG and Caliper instrumentation overheads for three instrumented applications.
- APPEKG overhead is near to 1%.
- Caliper reports low overhead with simple reporting, but extremely high overhead with more detailed reporting.



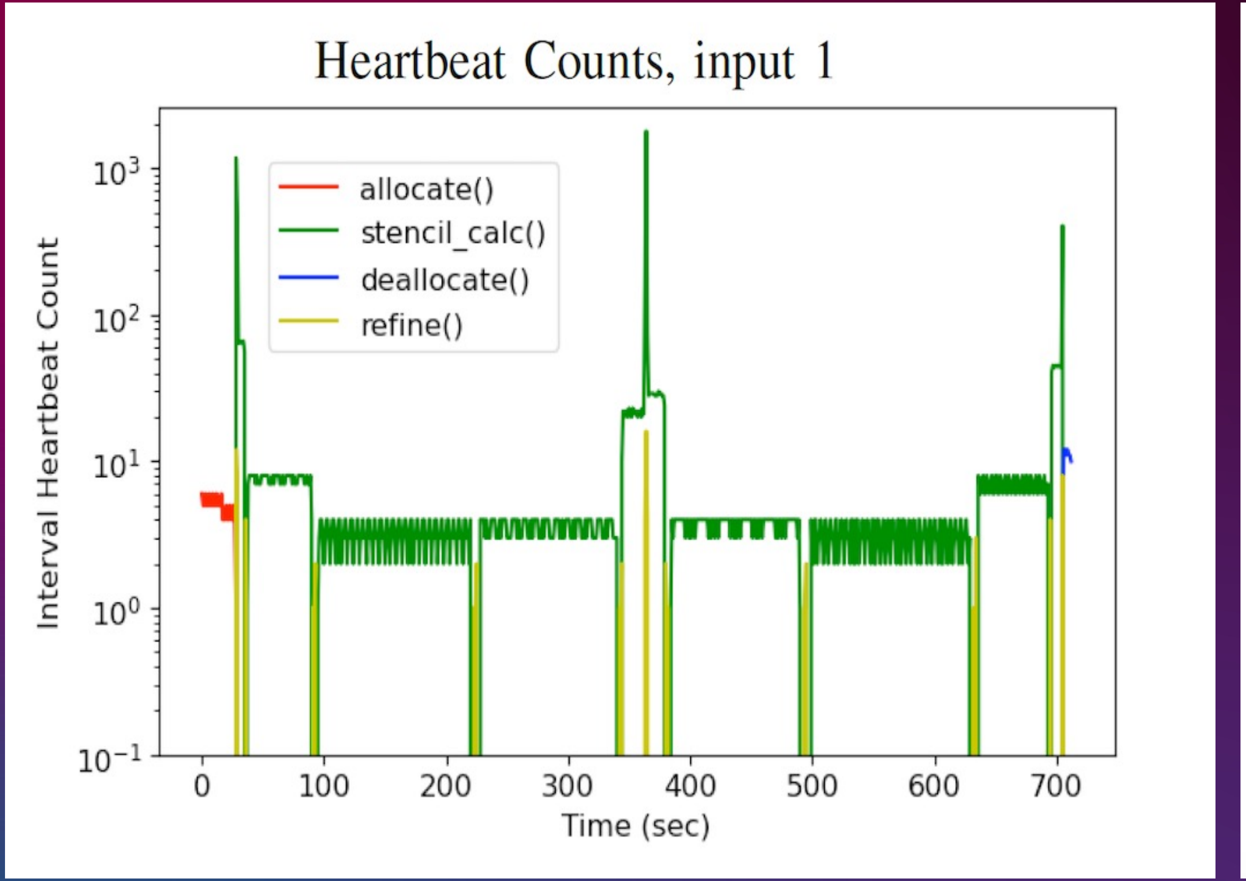
APPEKG AND CALIPER OVERHEADS.

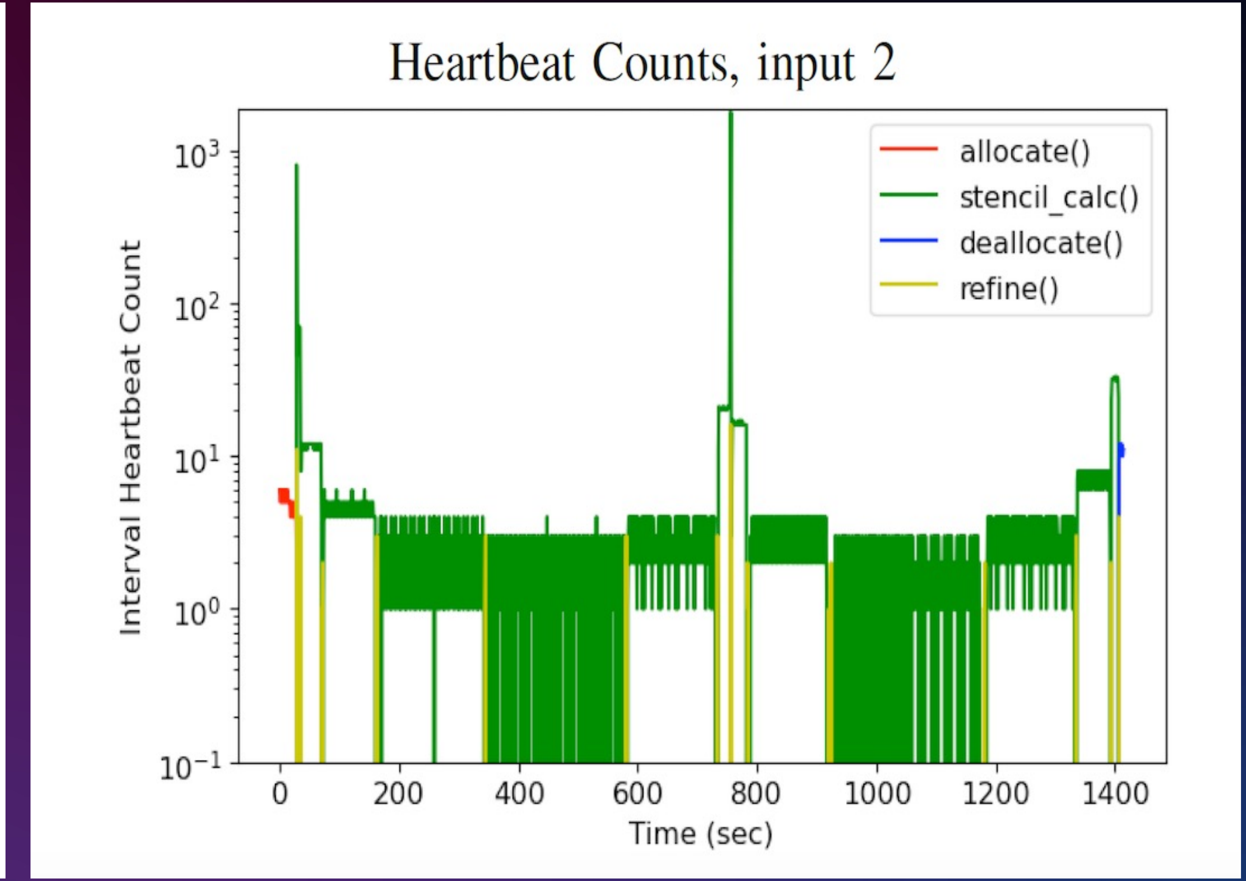
App	Uninst time (sec)	AppEKG overhead	Max Rate Factor	Caliper Overhead	
				Summary	Detailed
LAMMPS	462	0.60%	500K	<1%	400-1200%
MiniAMR	720	0.55%	100	~1%	n/a
MiniFE	844	1.18%	2M	<1%	40-350%

Heartbeat Analyses

- AppEKG is still in very early exploratory development, we do not have large and sophisticated heartbeat analyses developed.
- AppEKG can be used to create historical heartbeat data of applications that can be utilized to build a ML module to:
  - Detect anomalous future runs that deviate from the normal pattern.

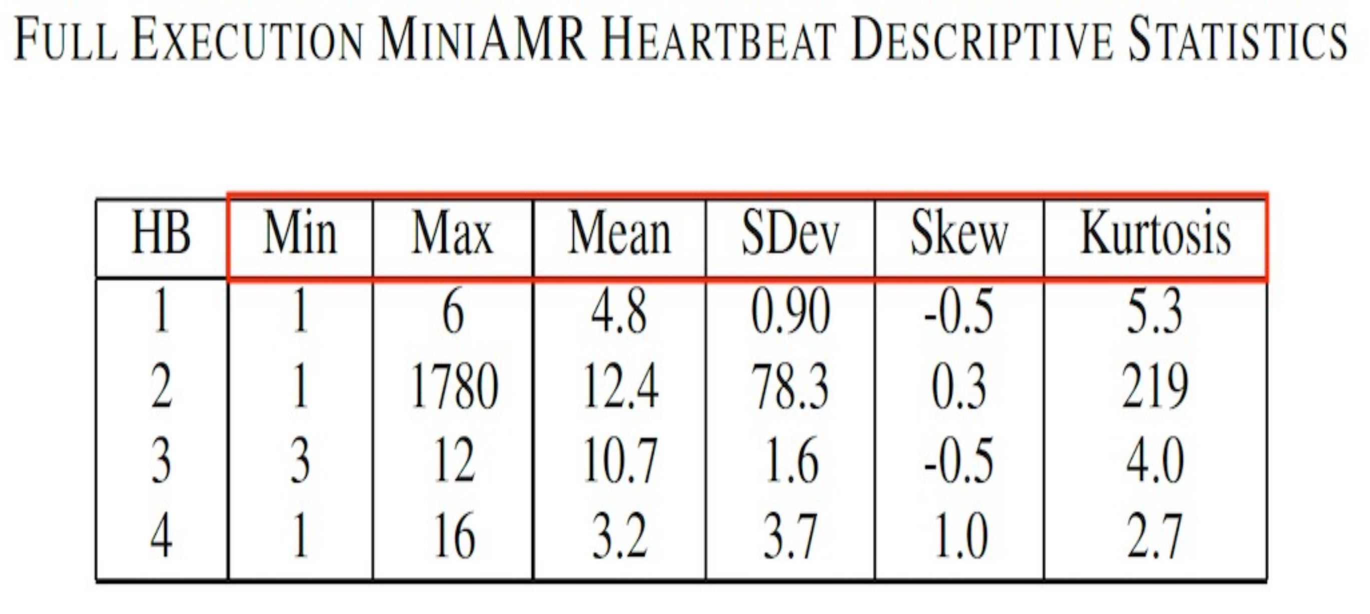
Example1: Heartbeat Data Presentations (Visual) MiniAMR





Example2: Heartbeat Data Presentations (Statistical) MiniAMR

- Descriptive statistics for each heartbeat counts of all processes.
- Such statistics can be also generated per process and per thread.
- Analyze the variance between processes or threads.

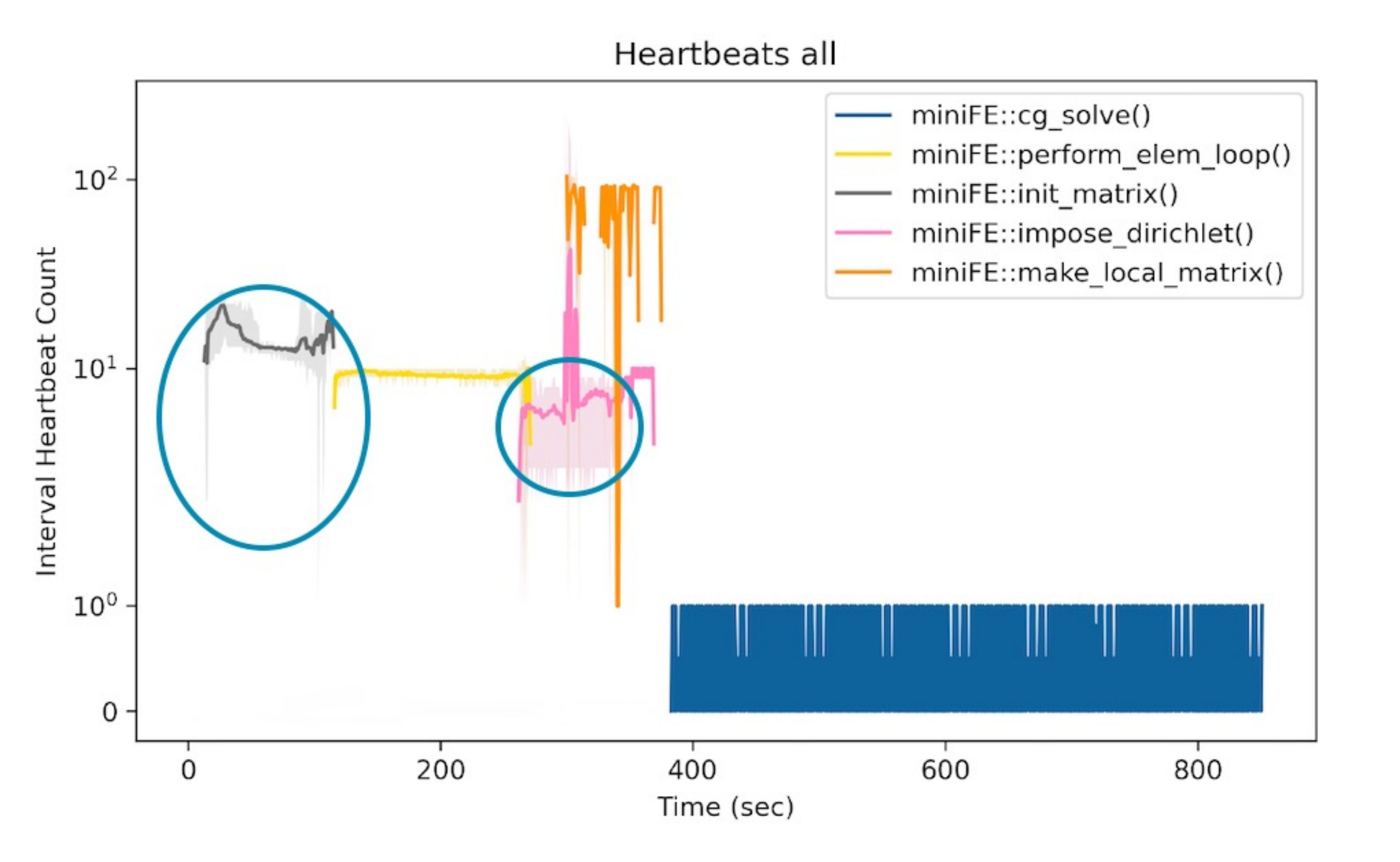


FULL EXECUTION MINIAMR HEARTBEAT DESCRIPTIVE STATISTICS

HB	Min	Max	Mean	SDev	Skew	Kurtosis
1	1	6	4.8	0.90	-0.5	5.3
2	1	1780	12.4	78.3	0.3	219
3	3	12	10.7	1.6	-0.5	4.0
4	1	16	3.2	3.7	1.0	2.7

Example3: Heartbeat Data Presentations (Statistical) MiniFE

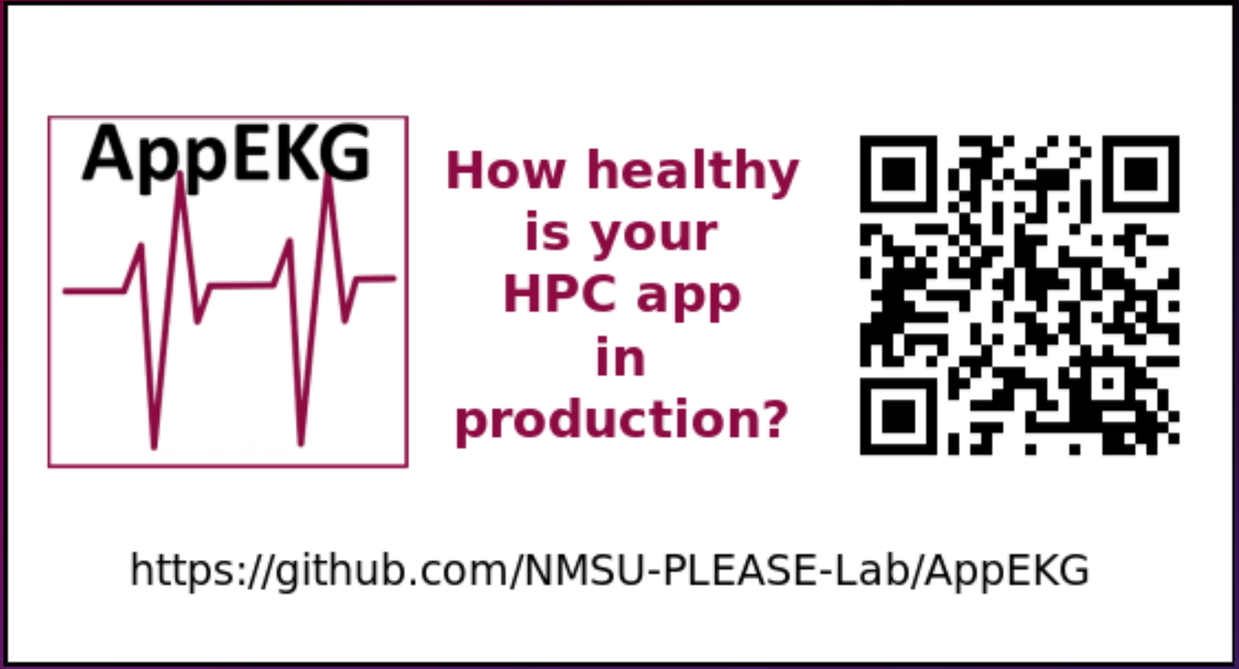
- Lines are all-processes average heartbeat count.
- Min/Max values form the shaded area around the average line.
- Min and Max values are extremely close to the average value.



Conclusion

- AppEKG is a novel approach to providing better insight into how HPC applications behave in production.
- It collects heartbeat data from most representative application phases.
- The main goal of AppEKG is to evaluating application performance in production.
- Many possible uses for heartbeat data – we are exploring some, would like to see others do so as well!

Thank You!



AppEKG

How healthy is your HPC app in production?

https://github.com/NMSU-PLEASE-Lab/AppEKG