The Effect of Asymmetric Performance on Asynchronous Task Based Runtimes

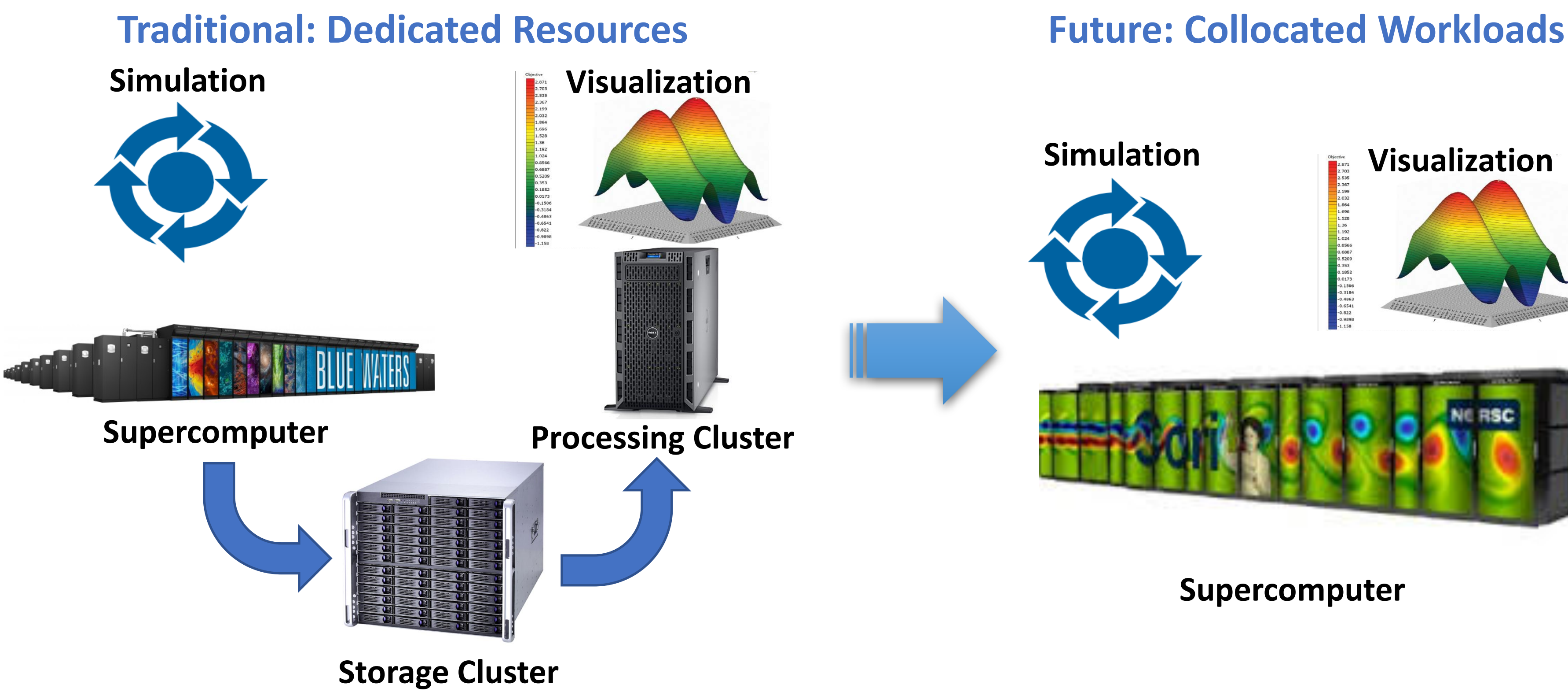
Debashis Ganguly and John R. Lange

The Prognostic Lab
Computer Science Department
University of Pittsburgh

ROSS 2017

Changing Face of HPC Environments

- Task-based Runtimes: Potential solution



Goal: Can asynchronous task-based runtimes handle ***asymmetric*** performance

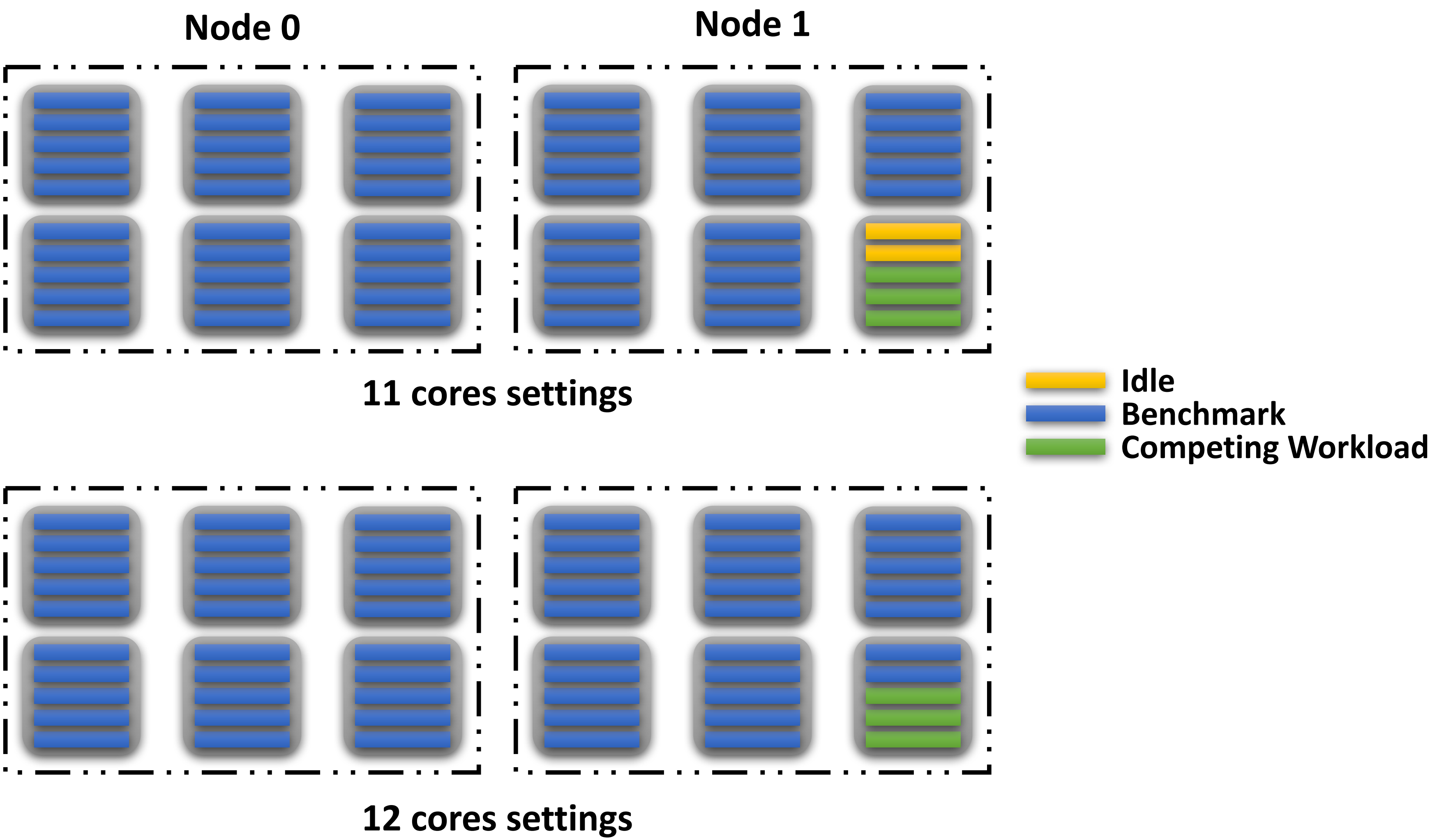
Task-based Runtimes

- Experiencing renewal in interest in systems community
  - Assumed to better address performance variability
- Adopt (Over-)Decomposed task-based model
  - Allow fine-grained scheduling decisions
  - Able to adapt to asymmetric/variable performance
- But…
  - Originally designed for application induced load imbalances, e.g., an adaptive mesh refinement (AMR) based application
  - Performance asymmetry can be of finer granularity, e.g., variable CPU time in time-shared environments

Basic Experimental Evaluation

- Synthetic situation
  - Emulate performance asymmetry in time-shared configuration
- Static and predictable setting
  - Benchmark on 12 cores, share one core with background workload
- Vary the percentage of CPU time of competing workload
  - Environment: 12 core dual socket compute node, *hyperthreading disabled*
  - Used ***cpulimit*** to control percentage of CPU time

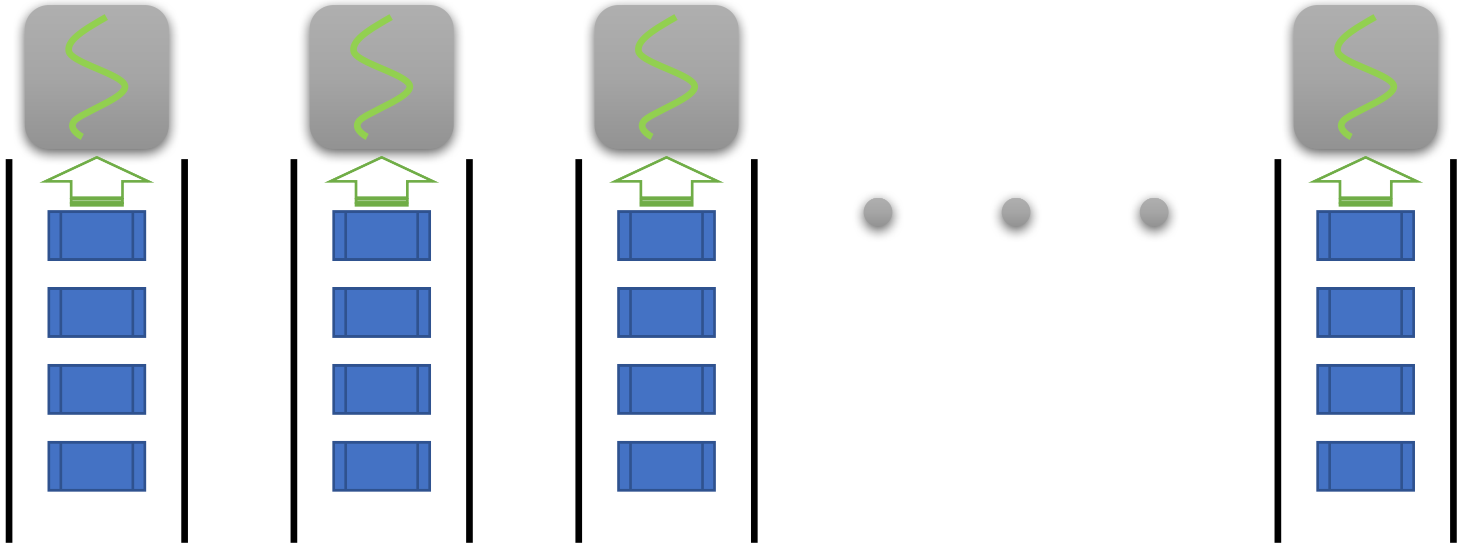
Workload Configuration



Experimental Setup

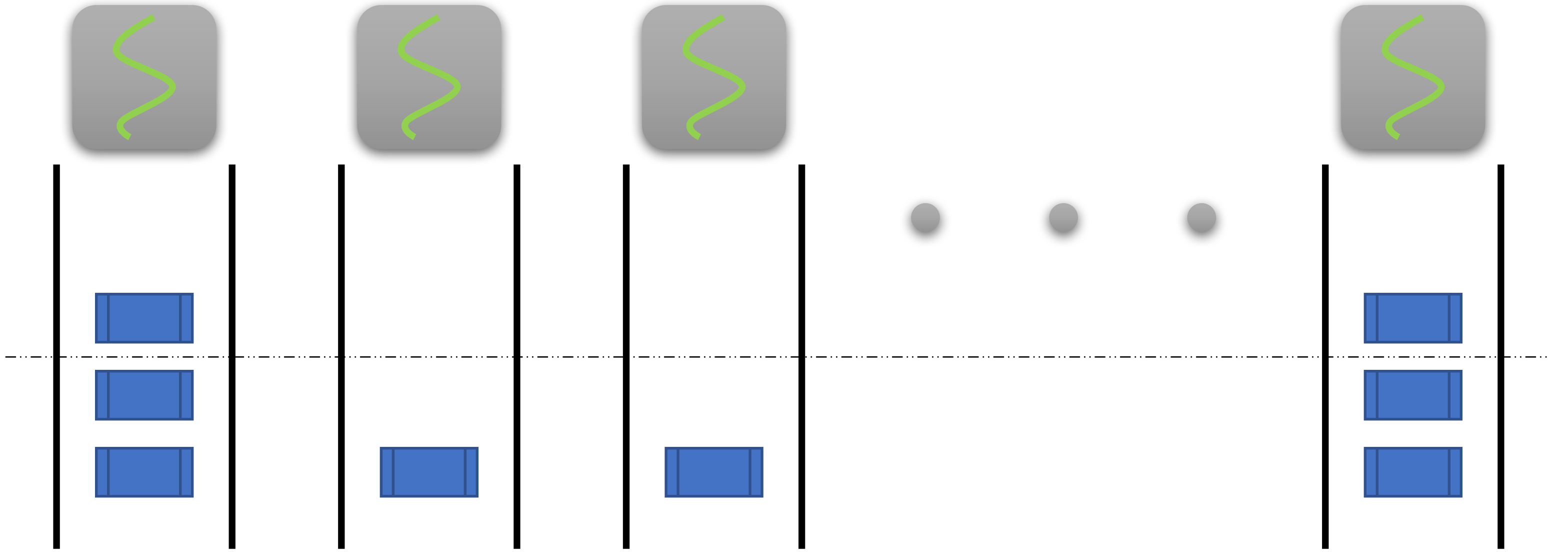
- **Evaluated two different runtimes:**
  - *Charm++*: LeanMD
  - *HPX-5*: LULESH, HPCG, LibPXGL
- **Competing Workload:**
  - *Prime Number Generator*: entirely CPU bound, a minimal memory footprint
  - *Kernel Compilation*: stresses internal OS features such as I/O and memory subsystem

Charm++



- Iterative over-decomposed applications
- Object based programming model
  - Tasks implemented as C++ objects
  - Objects can migrate across intra and inter-node boundaries

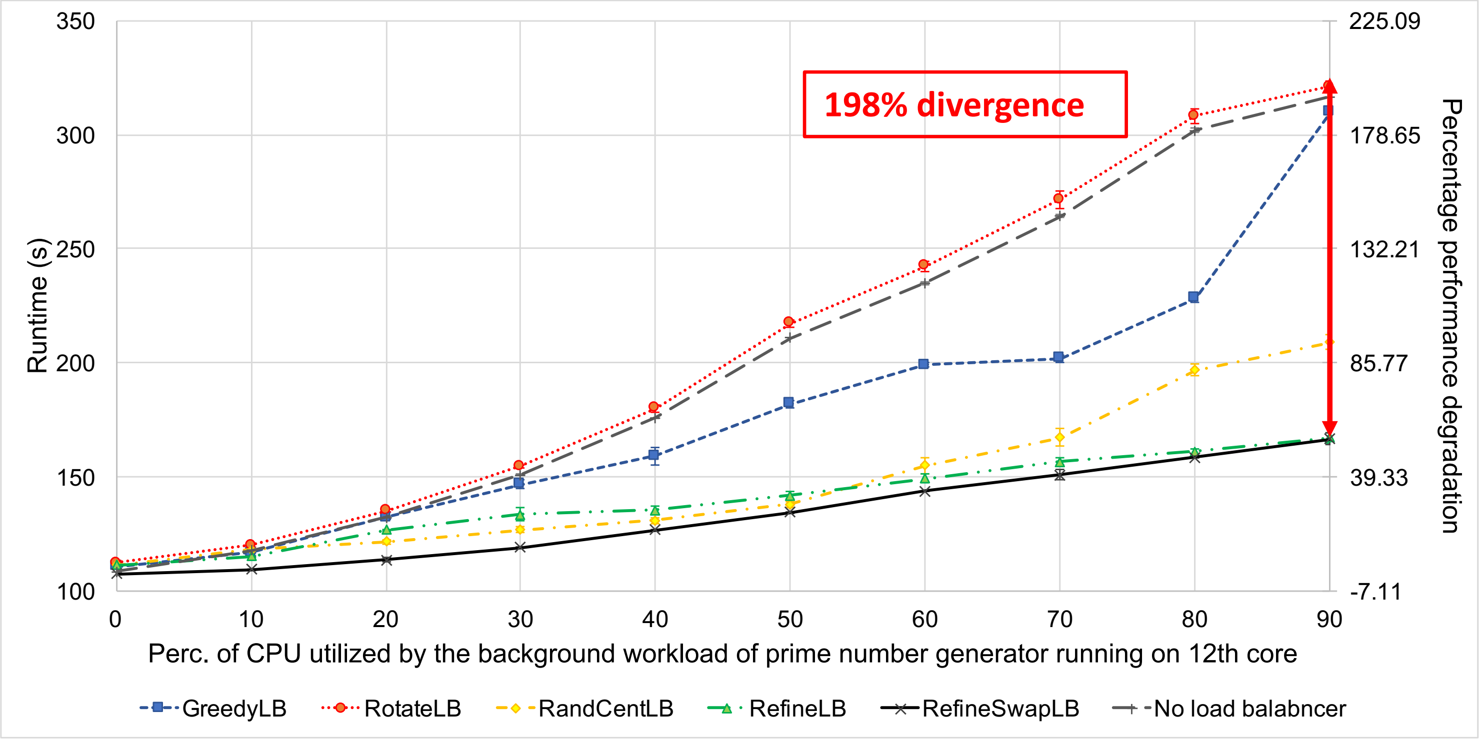
Charm++



- A separate *centralized* load balancer component
  - Preempts application progress
- Actively migrates objects based on current state
- Causes computation to block across the other cores

Choice of Load Balancer Matters

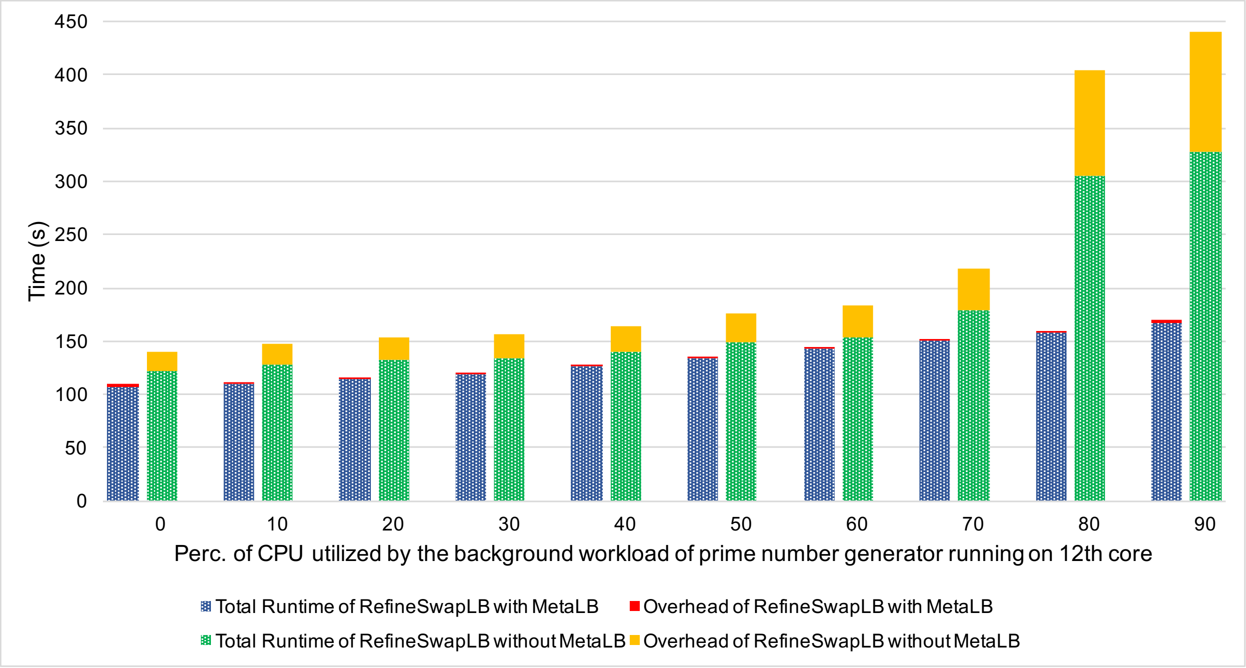
- Comparing performance of different load balancing strategies and without any load balancer



We selected RefineSwapLB for the rest of the experiments.

Invocation Frequency Matters

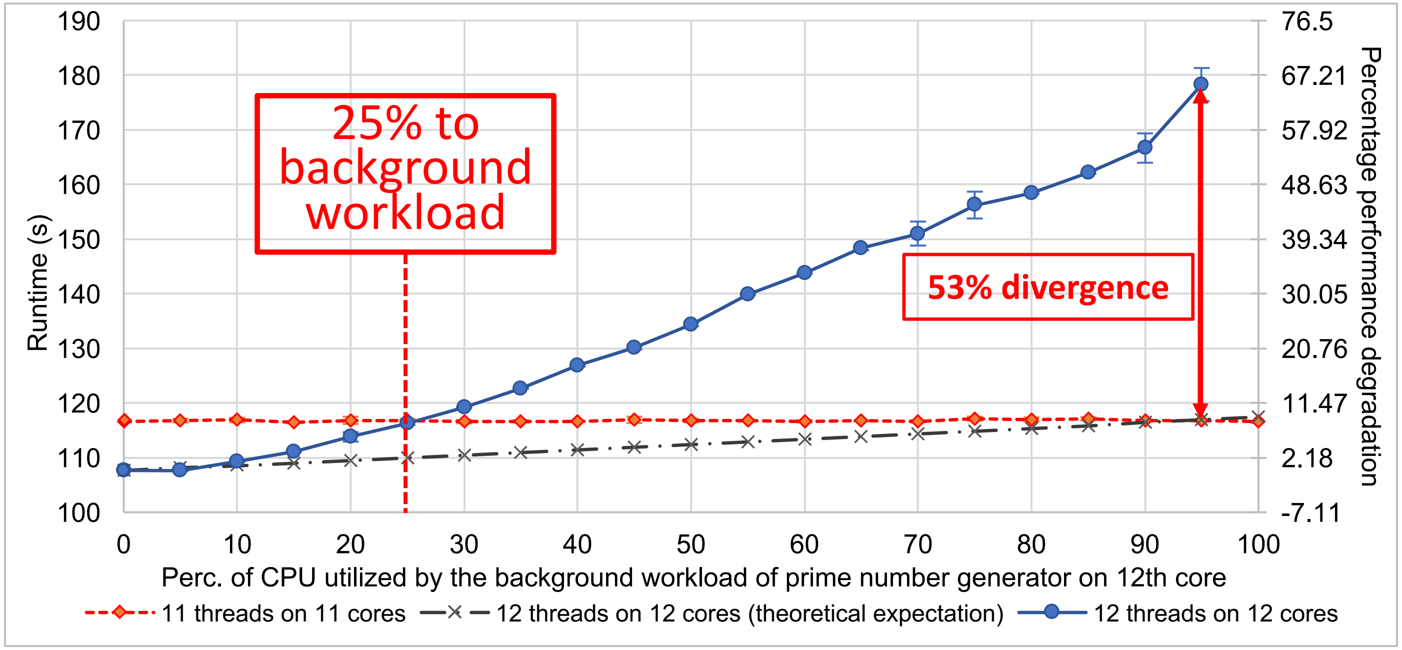
- MetaLB:
  - Invoke load balancer less frequently based on heuristics



Load balancing overhead of RefineSwapLB with or without MetaLB

We enabled MetaLB for our experiments.

Charm++: LEANMD

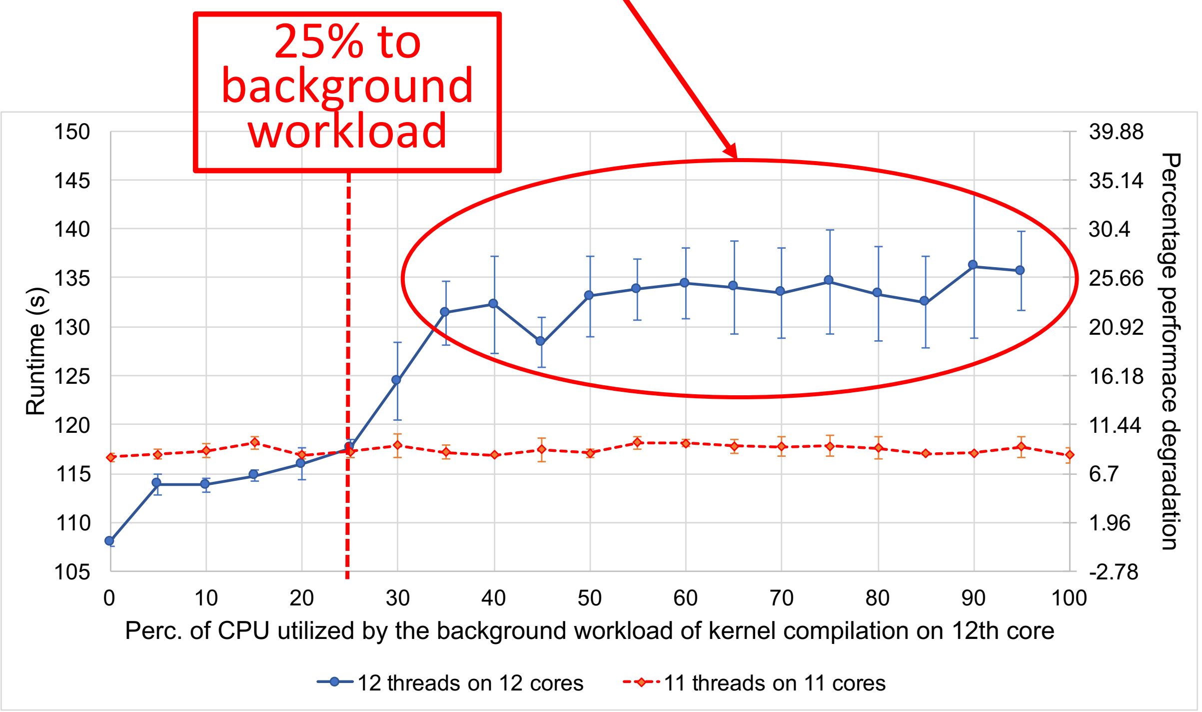


Sensitivity of perc. of CPU utilization by the background workload of prime number generator

- **12 cores are worse than 11 cores**
  - **…unless you have at least 75% of the core’s capacity.**
- If the application ***cannot get more than 75%*** of the core’s capacity, then is ***better off ignoring*** the core completely.

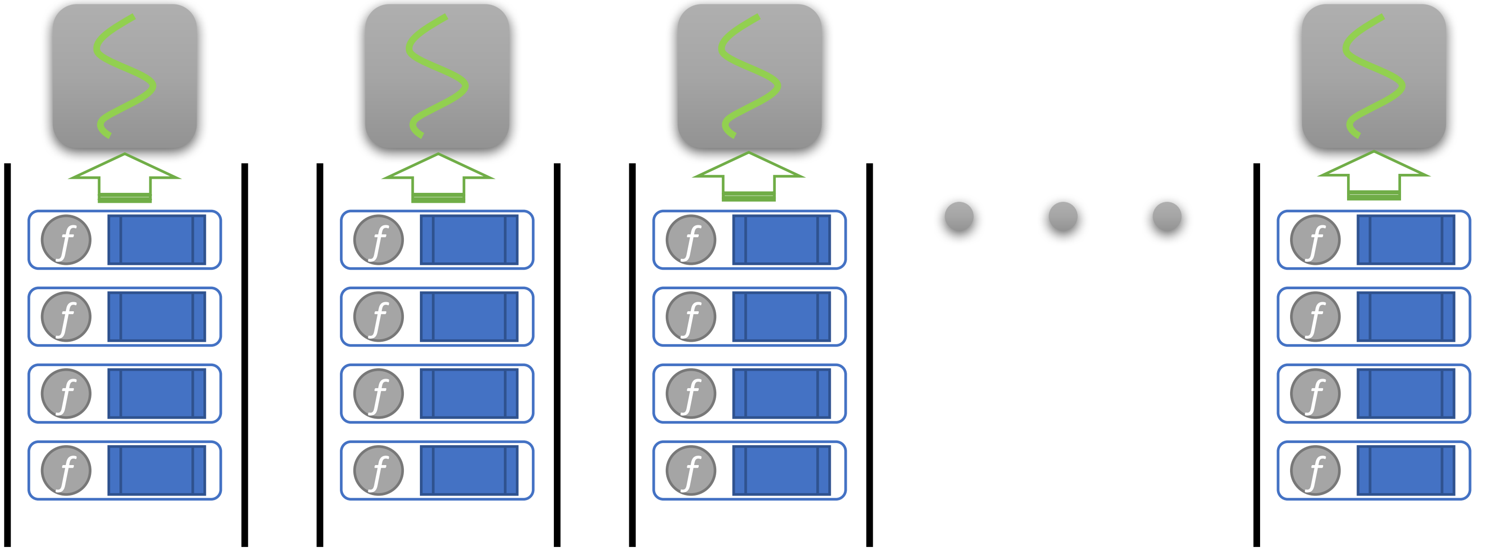
Charm++: LEANMD

More variable, but consistent mean performance.



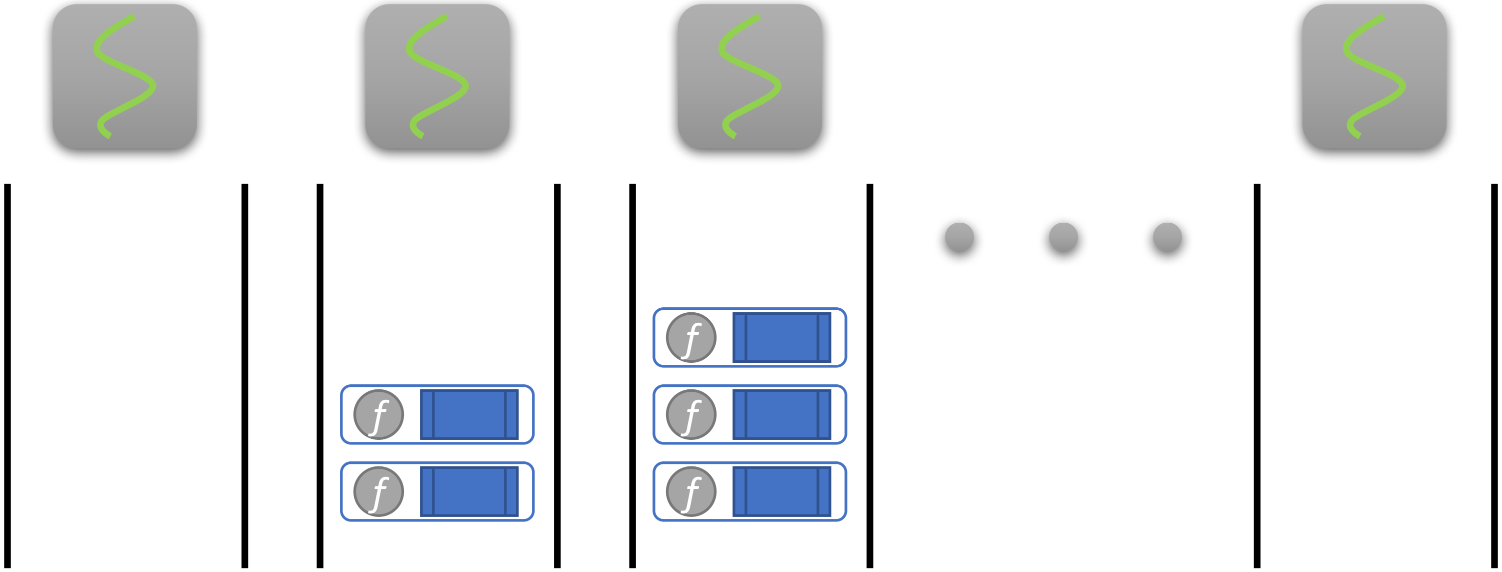
Sensitivity of perc. of CPU utilization by the background workload of kernel compilation

HPX-5



- Parcel:
  - Contains a computational task and a reference to the data the task operates on
- Follows ***Work-First*** principle of Cilk-5.
  - Every scheduling entity processes parcels from top of their scheduling queues.

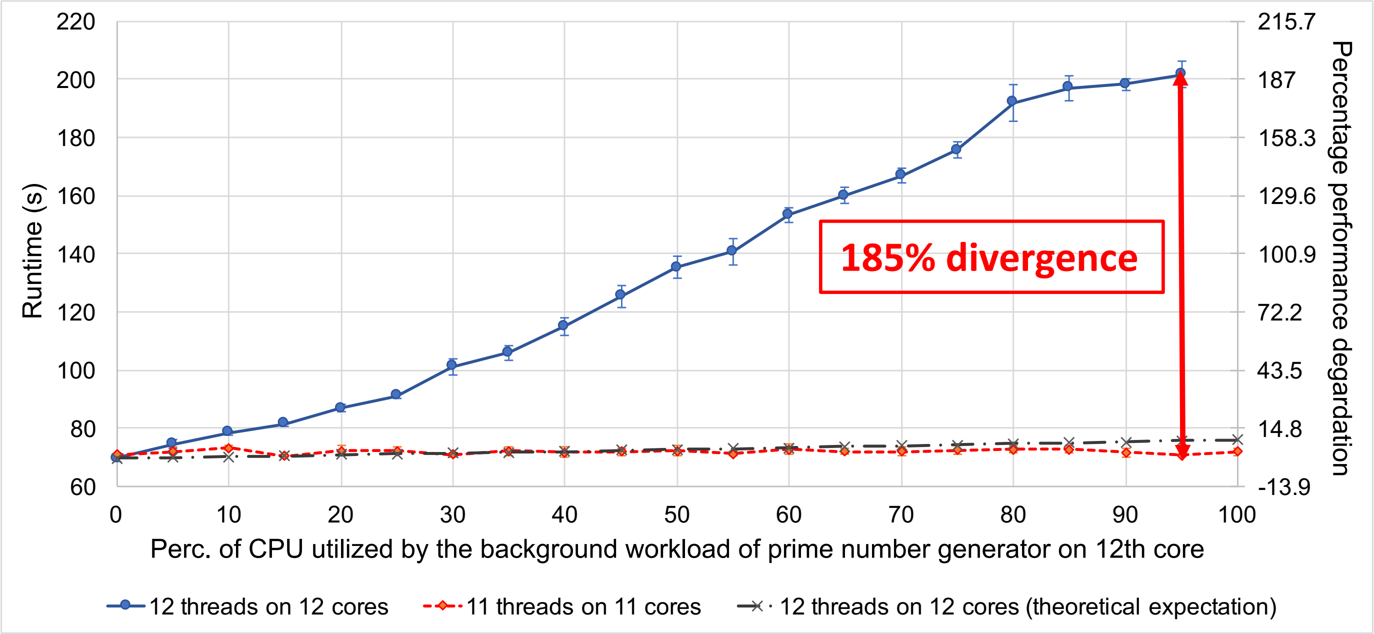
HPX-5



- Implemented using ***Random Work Stealing***
- ***No centralized decision making process***
- Overhead of work stealing is assumed by the stealer.

OpenMP: LULESH

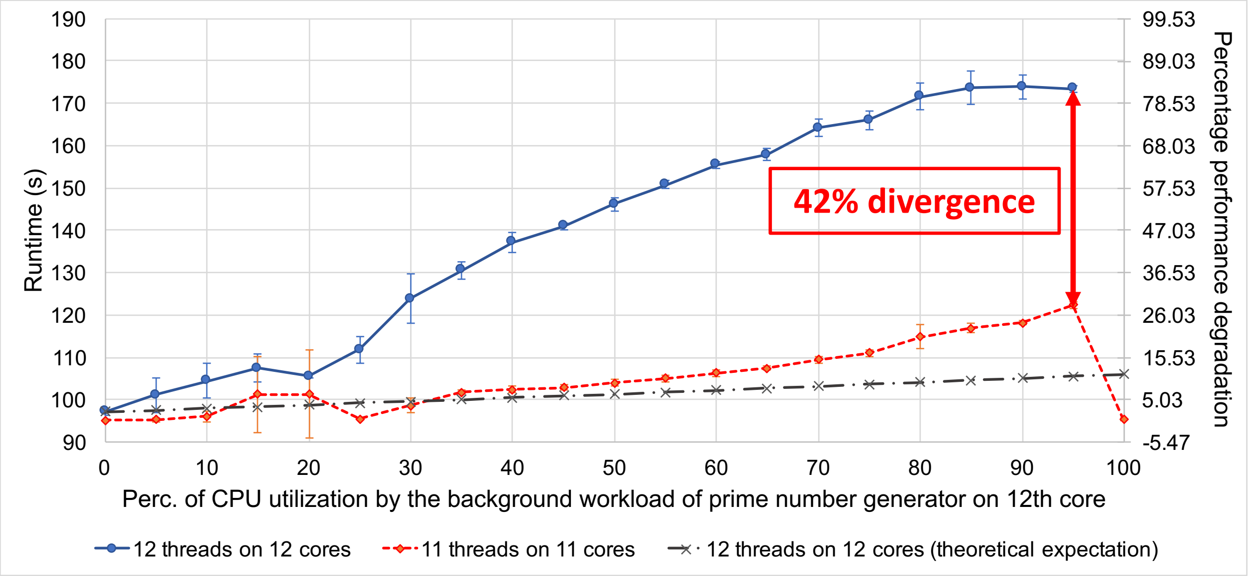
- Overall application performance determined by the slowest rank.
- Vulnerable to asymmetries in performance.
  - Rely on collective based communication.



Sensitivity of perc. of CPU utilization by the background workload of prime number generator

HPX-5: LULESH

- A traditional BSP application implemented using task-based programming

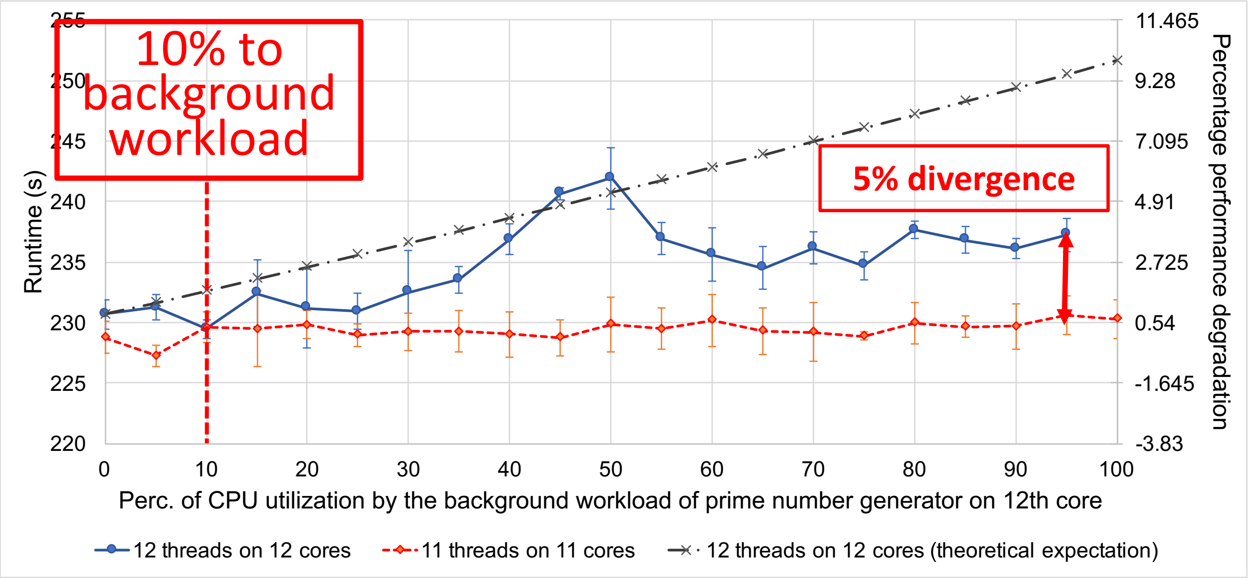


Sensitivity of perc. of CPU utilization by the background workload of prime number generator

- No cross-over point
- 12 cores are consistently worse than 11 cores

HPX-5: HPCG

- Another BSP application implemented in task-based model

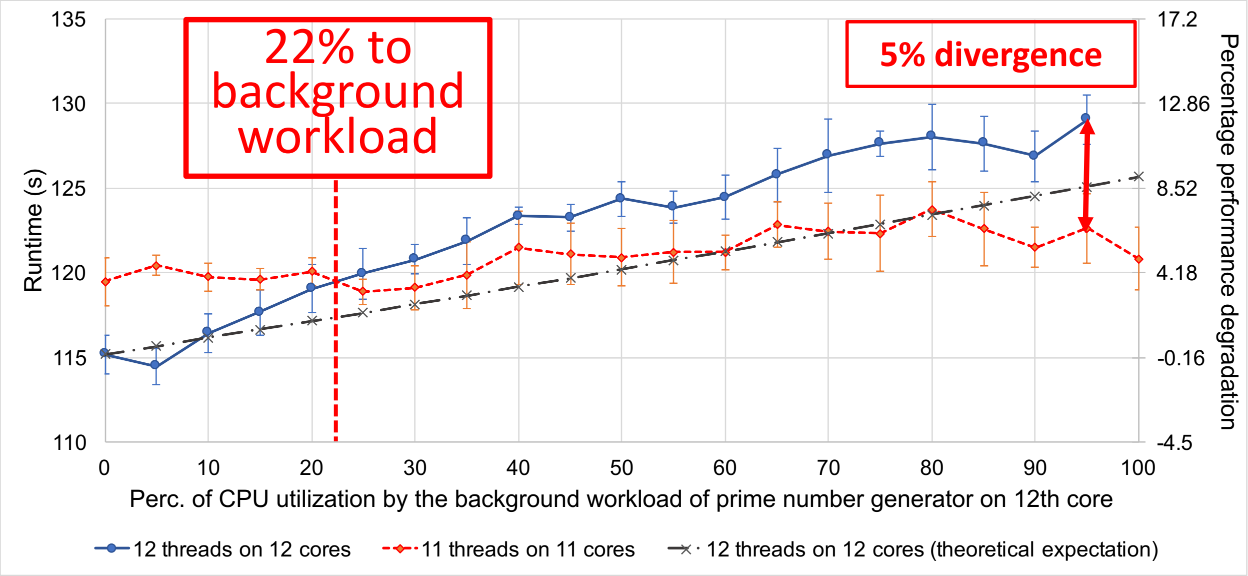


Sensitivity of perc. of CPU utilization by the background workload of prime number generator

- Better than the theoretical expectation
- 12 cores are consistently worse than 11 cores

HPX-5: LibPXGL

- An asynchronous graph processing library
  - A more natural fit

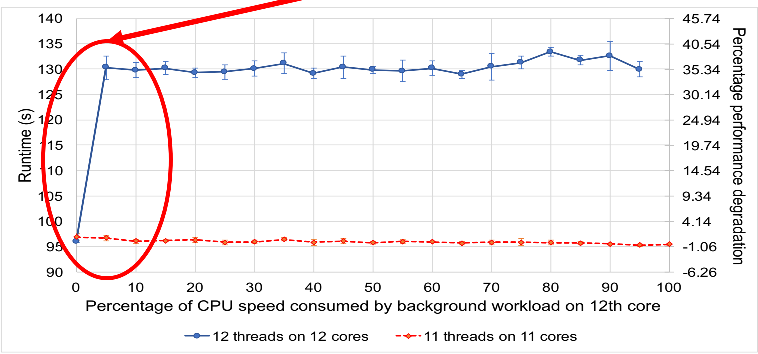


Sensitivity of perc. of CPU utilization by the background workload of prime number generator

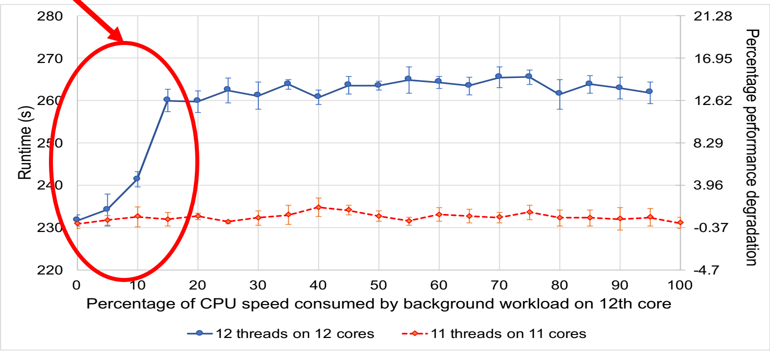
- No cross-over point
- 12 cores are consistently worse than 11 cores

HPX-5: Kernel Compilation

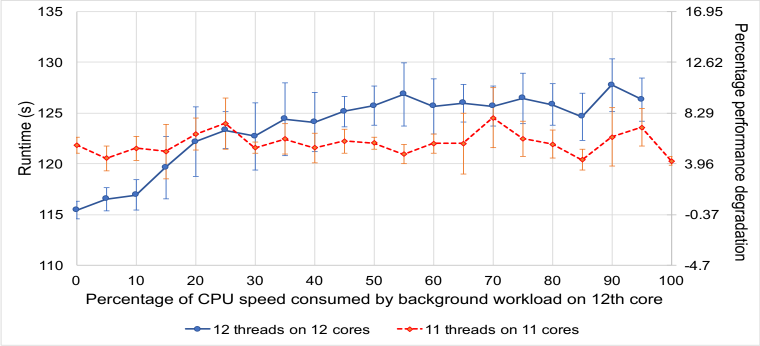
More immediate, instead of gradual decline.



LULESH



HPCG



LibPXGL

Conclusion

- **Performance asymmetry is still challenging**
- Preliminary evaluation:
  - Tightly controlled time-shared CPUs
  - Static and consistent configuration
- Better than BSP, but…
  - On average a CPU loses its utility to a task based runtime as soon as its performance diverges by only 25%.

Thank You

- Debashis Ganguly
  - Ph.D. Student, Computer Science Department, University of Pittsburgh
  - debashis@cs.pitt.edu
  - https://people.cs.pitt.edu/~debashis/
- The Prognostic Lab
  - http://www.prognosticlab.org