An Autonomic Performance Environment for Exascale

Kevin A. Huck[1], Nick Chaimov[1], Allen D. Malony[1], Sameer Shende[1], Allan Porterfield[2], Rob Fowler[2]

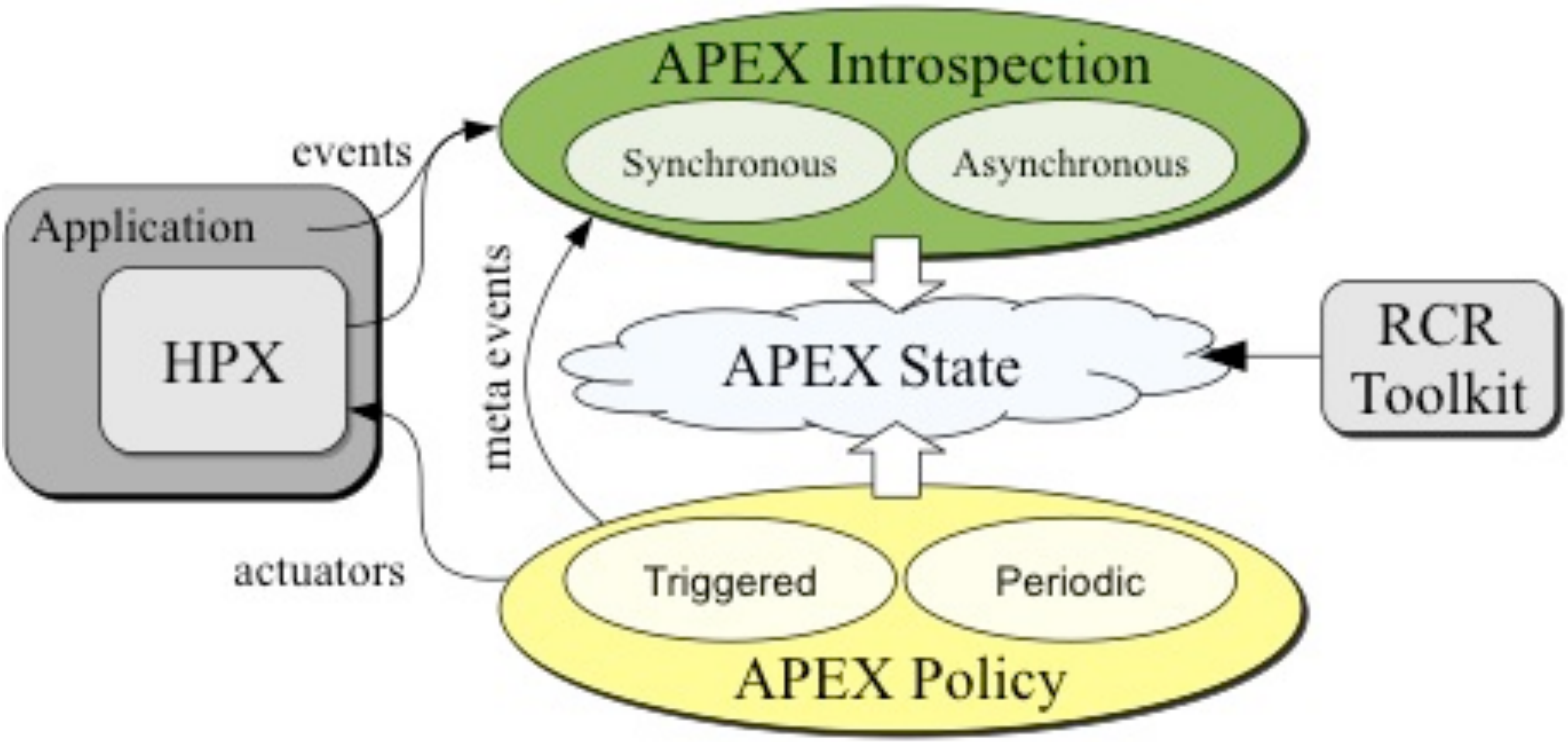
[1]University of Oregon, Eugene, Oregon, USA

[2]RENCI, Chapel Hill, North Carolina, USA

APEX and Autonomics

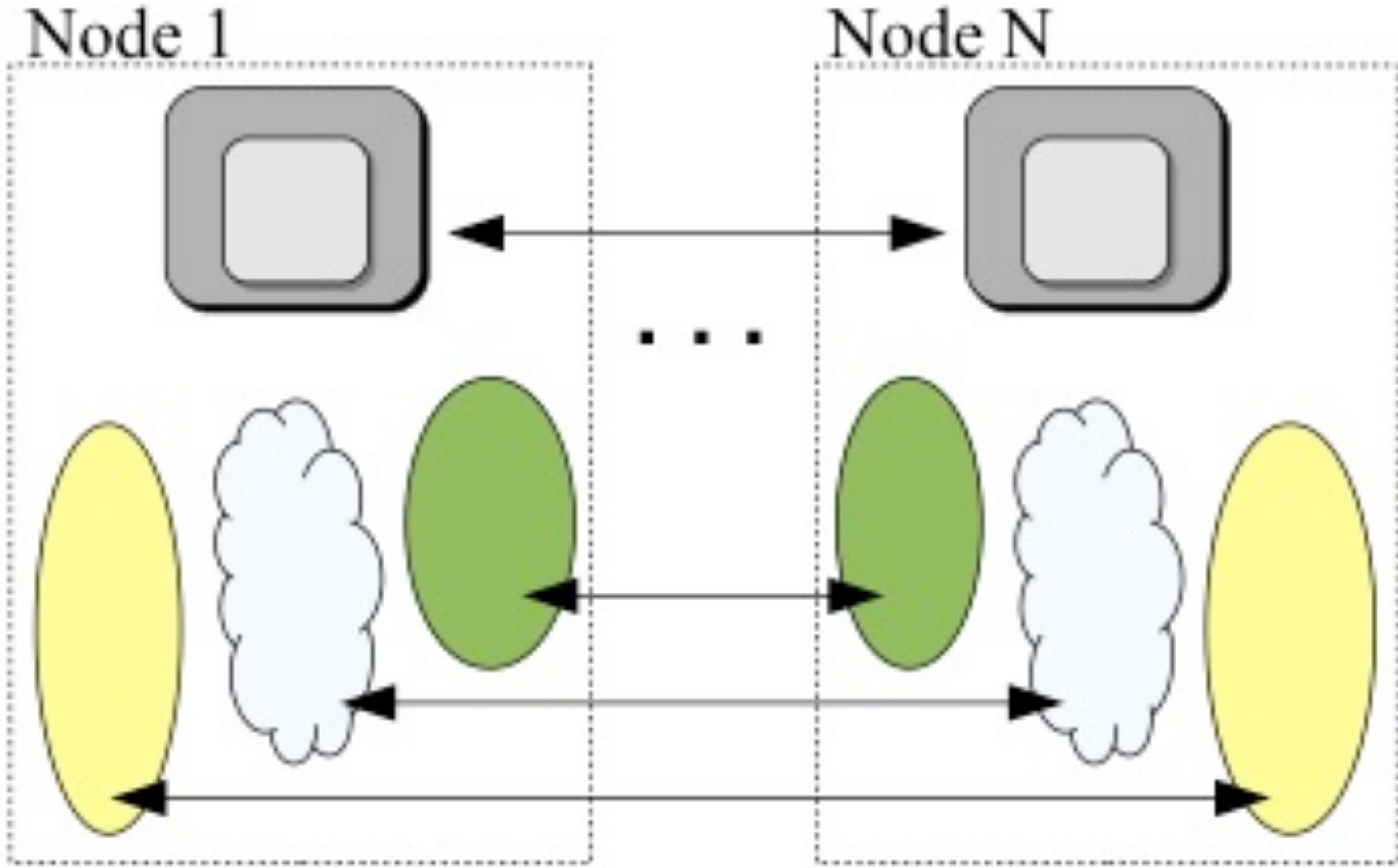
- Performance awareness and performance adaptation
- Top down and bottom up performance mapping / feedback
  - Make node-wide resource utilization data and analysis, energy consumption, and health information available in real time
  - Associate performance state with policy for feedback control
- APEX introspection
  - OS (LXK) track system resource assignment, utilization, job contention, overhead
  - Runtime (HPX) track threads, queues, concurrency, remote operations, parcels, memory management
  - ParalleX, DSLs and legacy codes allow language-level performance semantics to be measured

APEX Design



APEX Introspection
Synchronous
Asynchronous
events
Application
HPX
meta events
APEX State
RCR
Toolkit
actuators
Triggered
Periodic
APEX Policy

APEX Global Design



Leverage HPX to provide global introspection, state, and policies

UNIVERSITY OF OREGON

APEX Introspection

- APEX collects data through "inspectors"
  - *Synchronous* uses an event API and event "listeners"
    - Initialize, terminate, new thread
    - Timer start, stop, yield, resume
    - Sampled value (counters from HPX-5, HPX-3)
    - Custom events (meta-events)
  - *Asynchonous* do not rely on events, but occur periodically
- APEX exploits access to performance data from lower stack components
  - Reading from the RCR blackboard (i.e., power, energy)
  - "Health" data through other interfaces (e.g., /proc/stat from current systems)

RCR: *Resource Centric Reflection*

- Performance introspection across layers to enable dynamic, adaptive operation and decision control
- Extends previous work on building decision support instrumentation (*RCRToolkit*) for introspective adaptive scheduling
- Daemon monitors shared, non-core resources
- Real-time analysis, raw/processed data published to shared memory region, clients subscribe
- Utilized at lower levels of the OpenX stack
- APEX introspection and policy components will access and evaluate

APEX Event Listeners

- Profiling listener
  - Start event: take timestamp, return profiler handle
  - Stop event: take timestamp, put profiler object in a queue for back-end processing, return
  - Sample event: put the sample in the queue
  - Consumer thread: process profiler objects and samples to build statistical profile (in HPX-3, processed as a thread/ task)
- Concurrency listener
  - Start event: push timer ID on stack
  - Stop event: pop timer ID off stack
  - Consumer thread: periodically log current timer for each thread, output report at termination

APEX Policy Listener

- Policies are rules that decide on outcomes based on observed state
  - *Triggered* policies are invoked by introspection API events
  - *Periodic* policies are run periodically
- Polices are registered with the Policy Engine
  - Applications, runtimes, and/or OS register callback functions
- Callback functions define the policy rules
  - "If x < y then…"
- Enables runtime adaptation using introspection data
  - Engages actuators across stack layers
  - Could also be used to involve online auto-tuning support

APEX Global View

- All APEX introspection is collected locally
  - However, it is not limited to a single-node view
- Global view of introspection data and interactions
  - Take advantage of the distributed runtime support
    - HPX3, HPX5, MPI, …
- API provided for back-end implementations
  - *apex_global_get_value()* – each node gets data to be reduced, optional RDMA put (push model)
  - *apex_global_reduce()* – optional RDMA get (pull model), node data is aggregated at root node, optional broadcast back out
- Can extend global view to policies

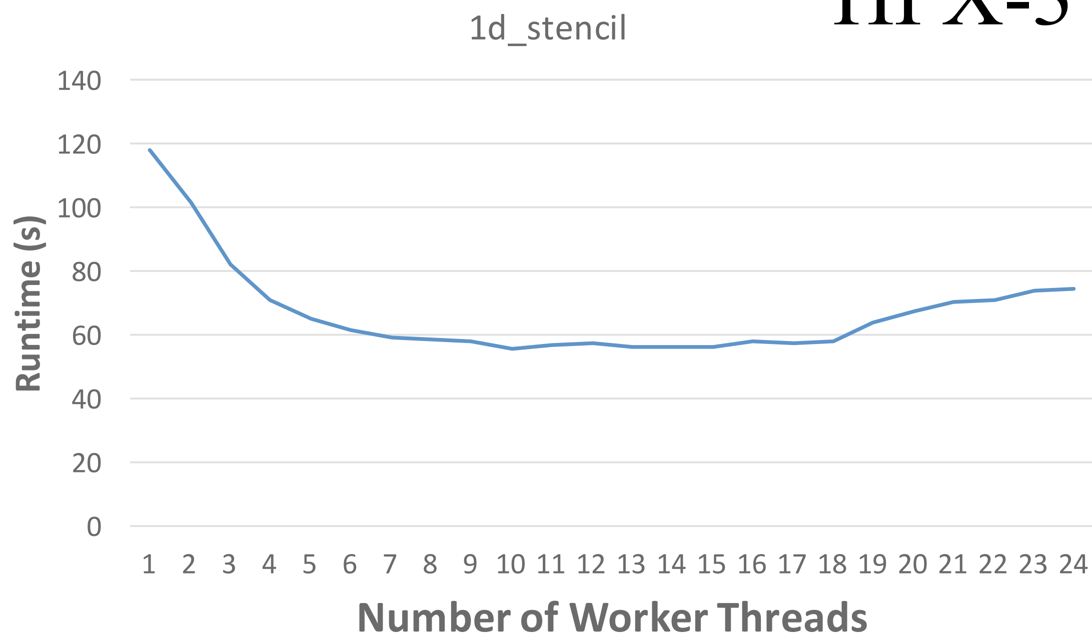
APEX Examples

- HPX-3 1-D stencil code
- HPX-5 Single-source-shortest-path benchmark
- HPX-5 LULESH kernel
- HPX-3 miniGhost kernel
- All experiments conducted on Edison
  - Cray XC30 @ NERSC.gov
  - 5576 nodes with two 12-core Intel "Ivy Bridge" processors at 2.4 GHz
  - 48 threads per node (24 physical cores w/hyperthreading)
  - Cray Aries interconnect with Dragonfly topology with 23.7 TB/s global bandwidth

Concurrency Throttling for Performance

- Heat diffusion
- 1D stencil code
- Data array partitioned into chunks
- 1 node with no hyperthreading

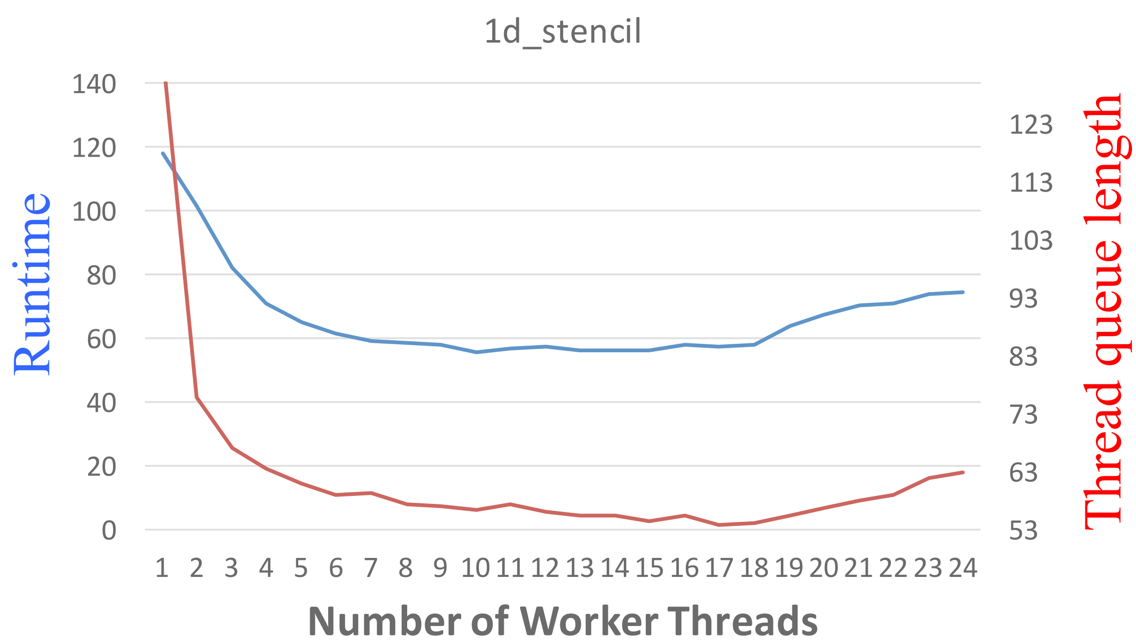
HPX-3



- Performance increases to a point with increasing worker threads, then decreases

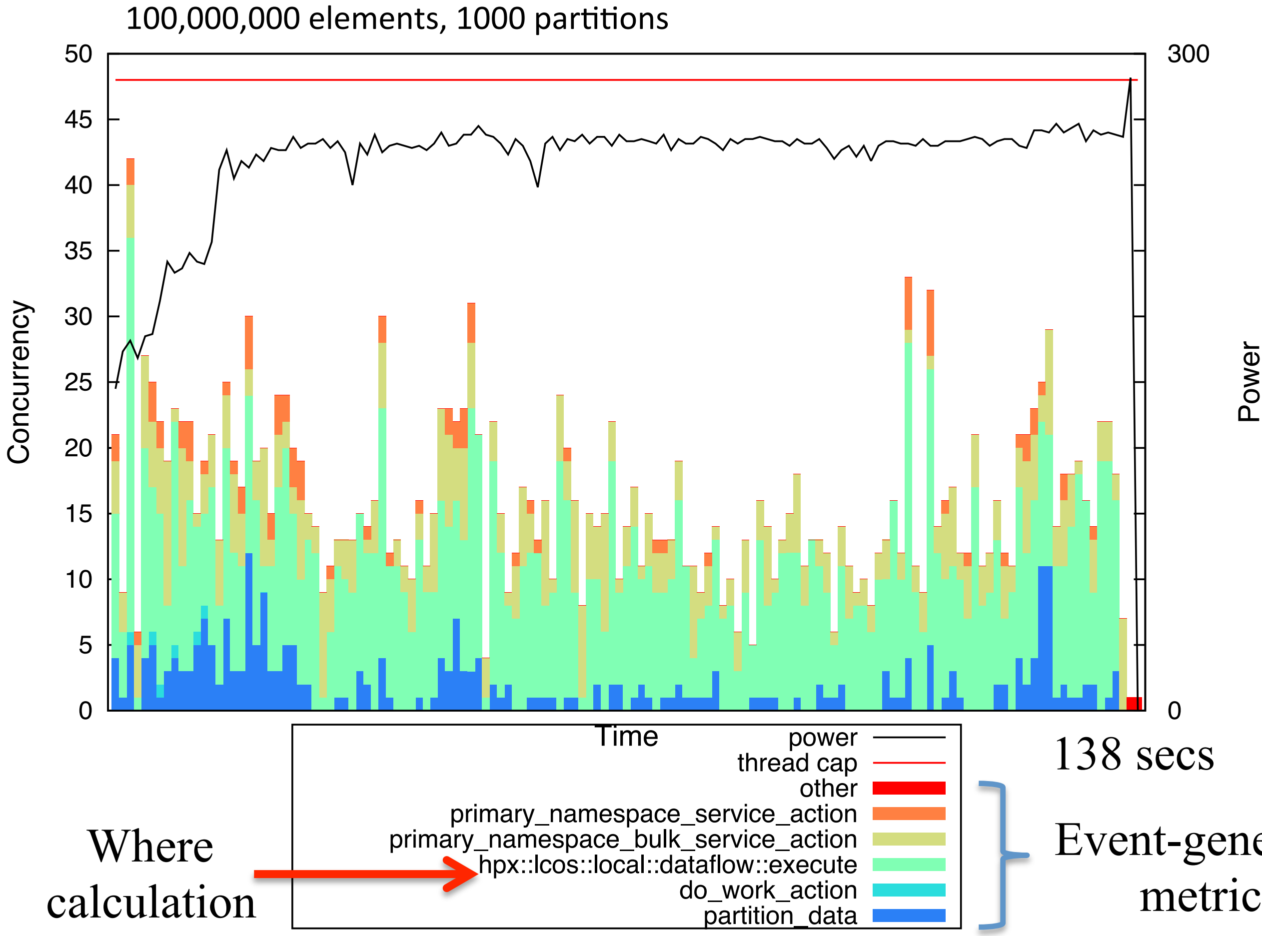
Concurrency Throttling for Performance

- Region of maximum performance correlates with thread queue length runtime performance counter
  – Represents # tasks currently waiting to execute



- Could do introspection on this to control concurrency throttling policy (*work in progress)

1d_stencil Baseline



138 secs

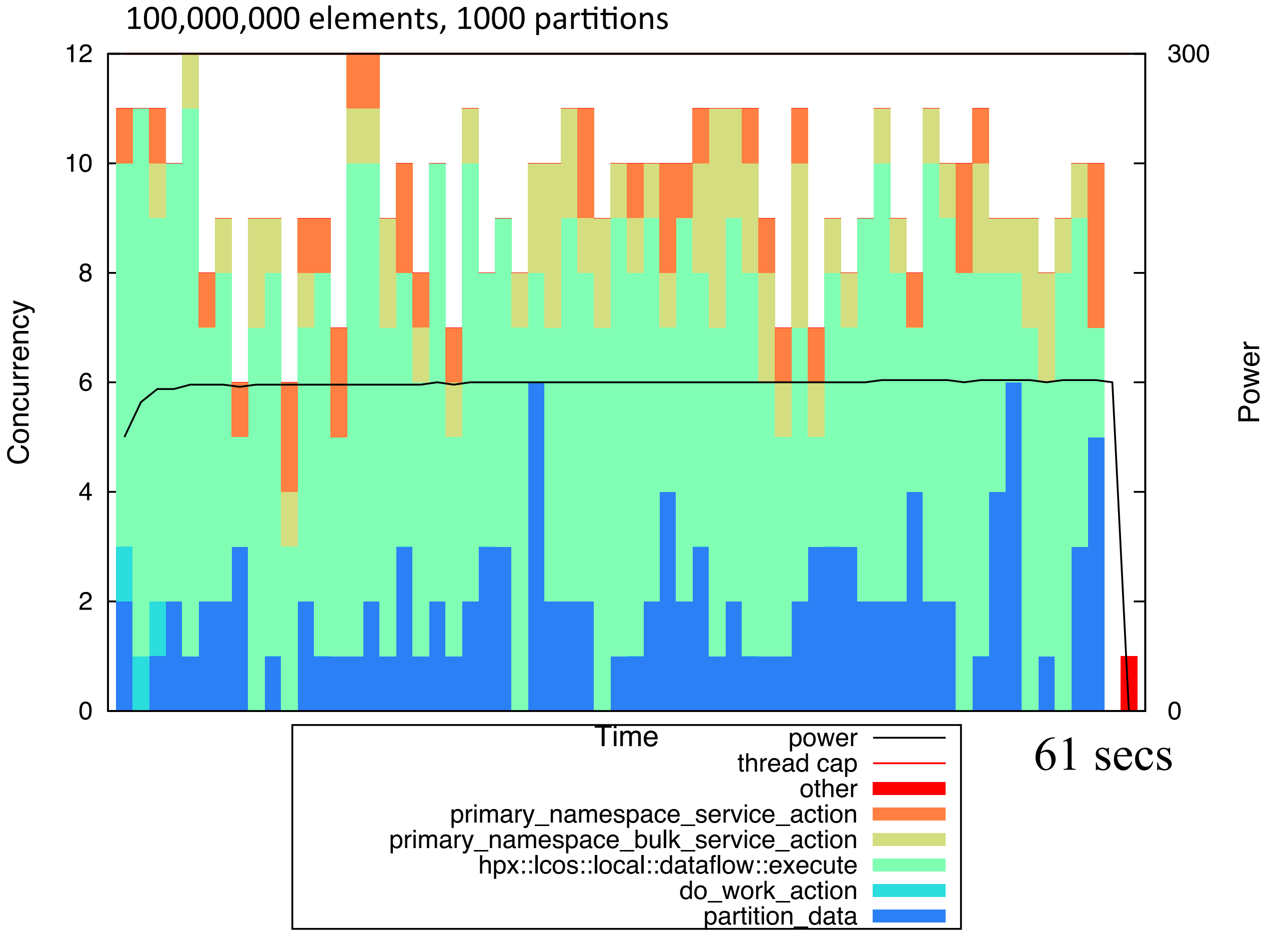
Event-generated metrics

Where calculation takes place

- 48 worker threads (with hyperthreading)
- Actual concurrency much lower
  - Implementation is memory bound
- Large variation in concurrency over time
  - Tasks waiting on prior tasks to complete

UNIVERSITY OF OREGON

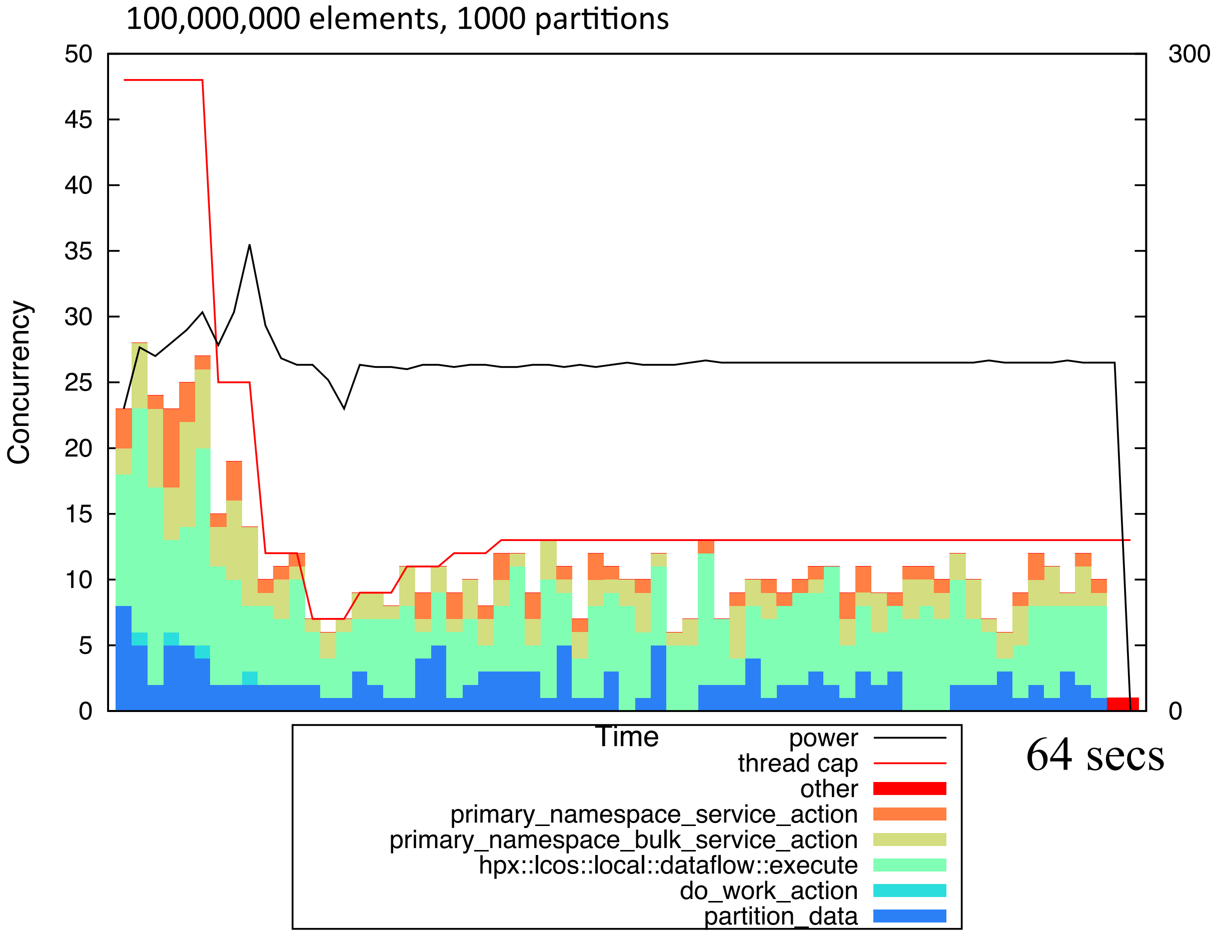
1d_stencil w/optimal # of Threads



61 secs

- 12 worker threads
- Greater proportion of threads kept busy
  - Less interference between active threads and threads waiting for memory
- Much faster
  - 61 sec. vs 138 sec.

1d_stencil Adaptation with APEX

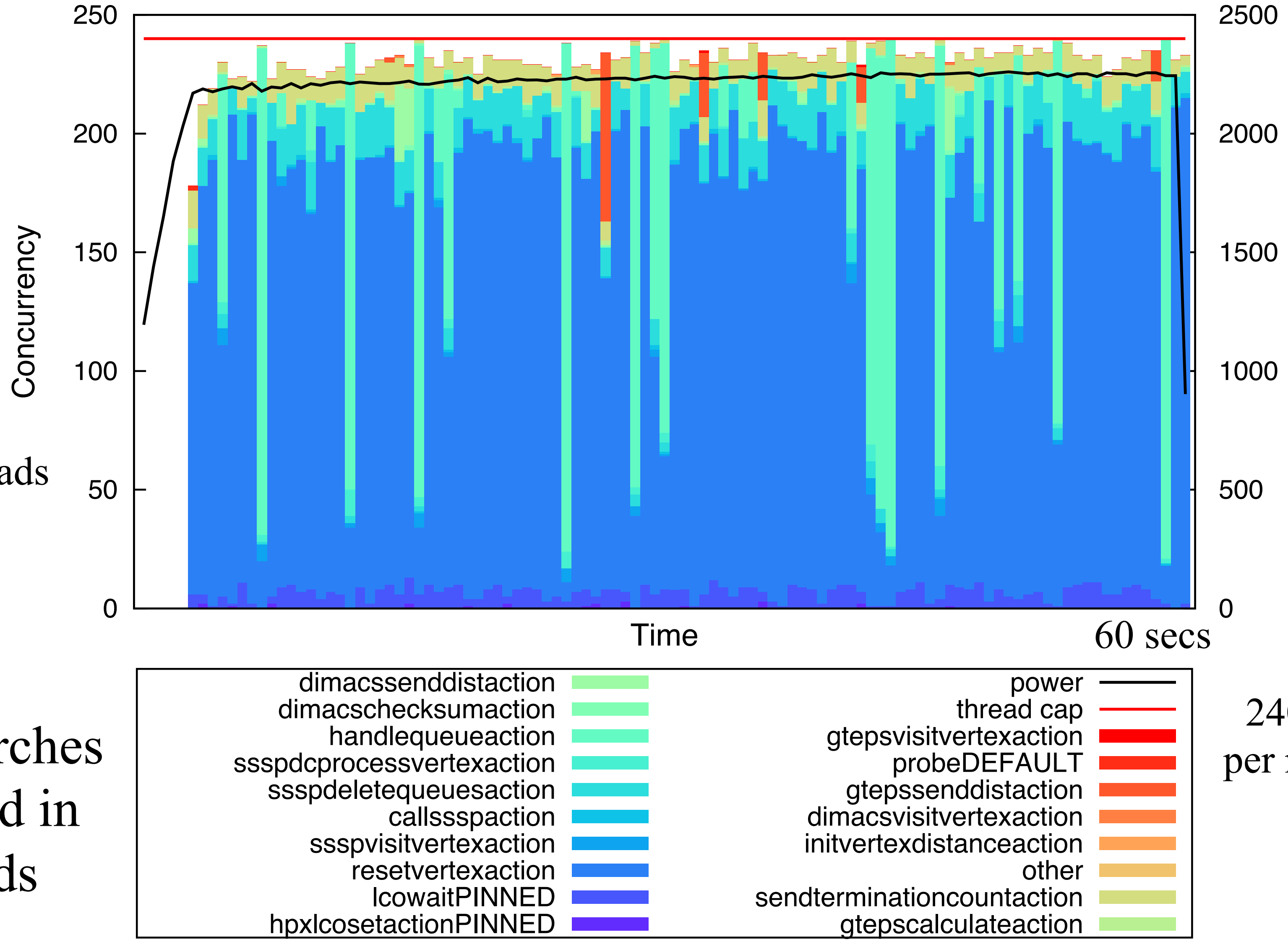


- Initially 48 worker threads
- Discrete hill climbing search to minimize average #of pending tasks
- Converges on 13 (vs. optimal of 12)
- Nearly as fast as optimal
  - 64 seconds vs. 61 seconds

Throughput Adaptation

- Single Source, Shortest Path (SSSP) graph search benchmark
  - Graph500.org benchmark kernel (http://www.graph500.org)
- Large graph loaded, a point is selected at random and the shortest path between it and all other points is found
  - Random4-n.10 dataset, runs for 60 seconds of timed searches
- Throughput is the metric of interest, not time to completion
- 10 nodes, 24 threads per node (no hyperthreading)
  - Graph is distributed across nodes
- APEX policy rule:
  - # calls to *_handle_queue_action()* used as "throughput" metric
  - Adjust thread concurrency to maximize throughput
  - Use Active Harmony for optimization search
    - *Parallel Rank Order* search strategy
- HPX-5 implementation

SSSP Baseline

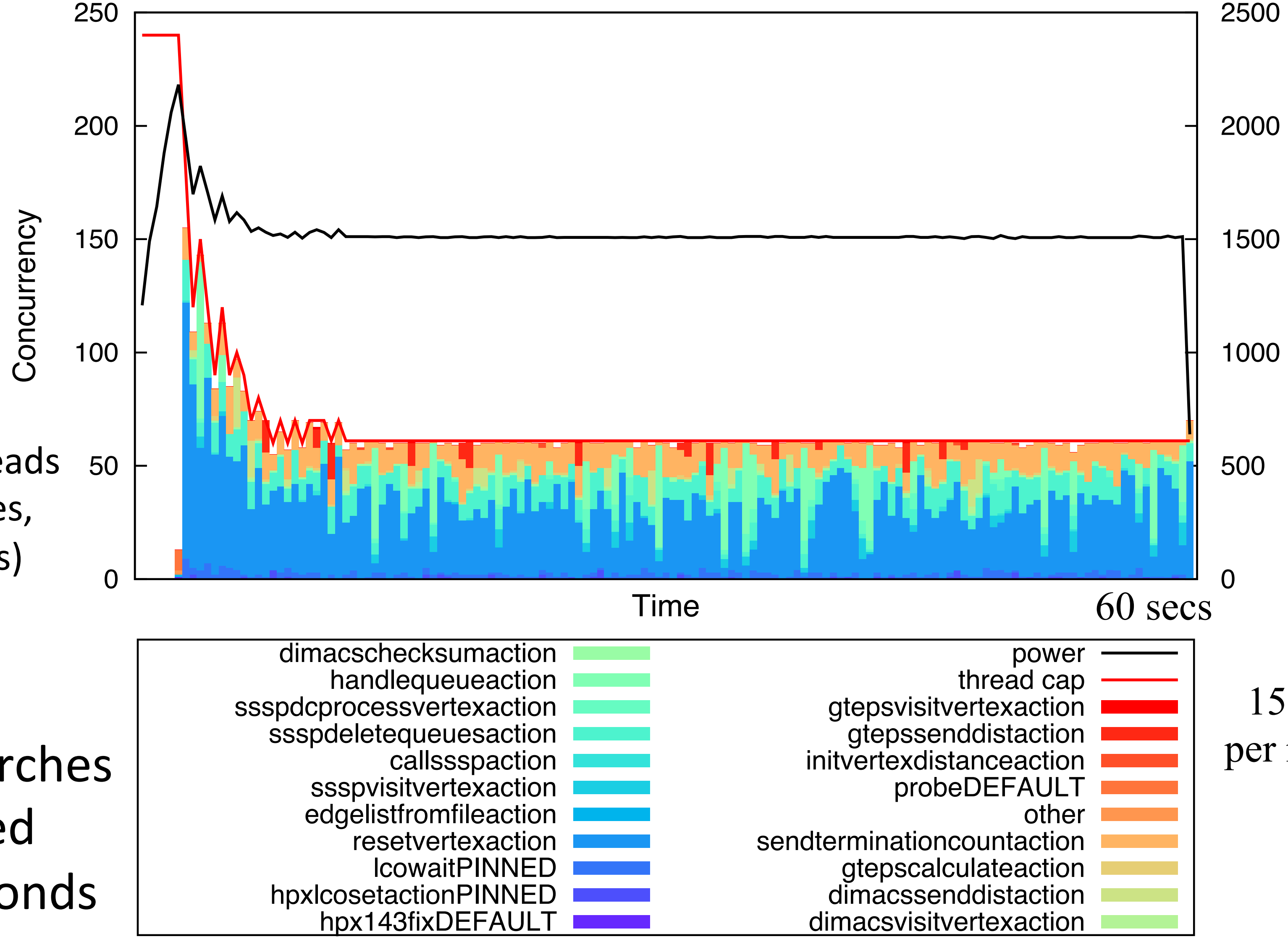


All 240 threads are busy

1962 searches performed in 60 seconds

240W per node

SSSP with Throughput Policy

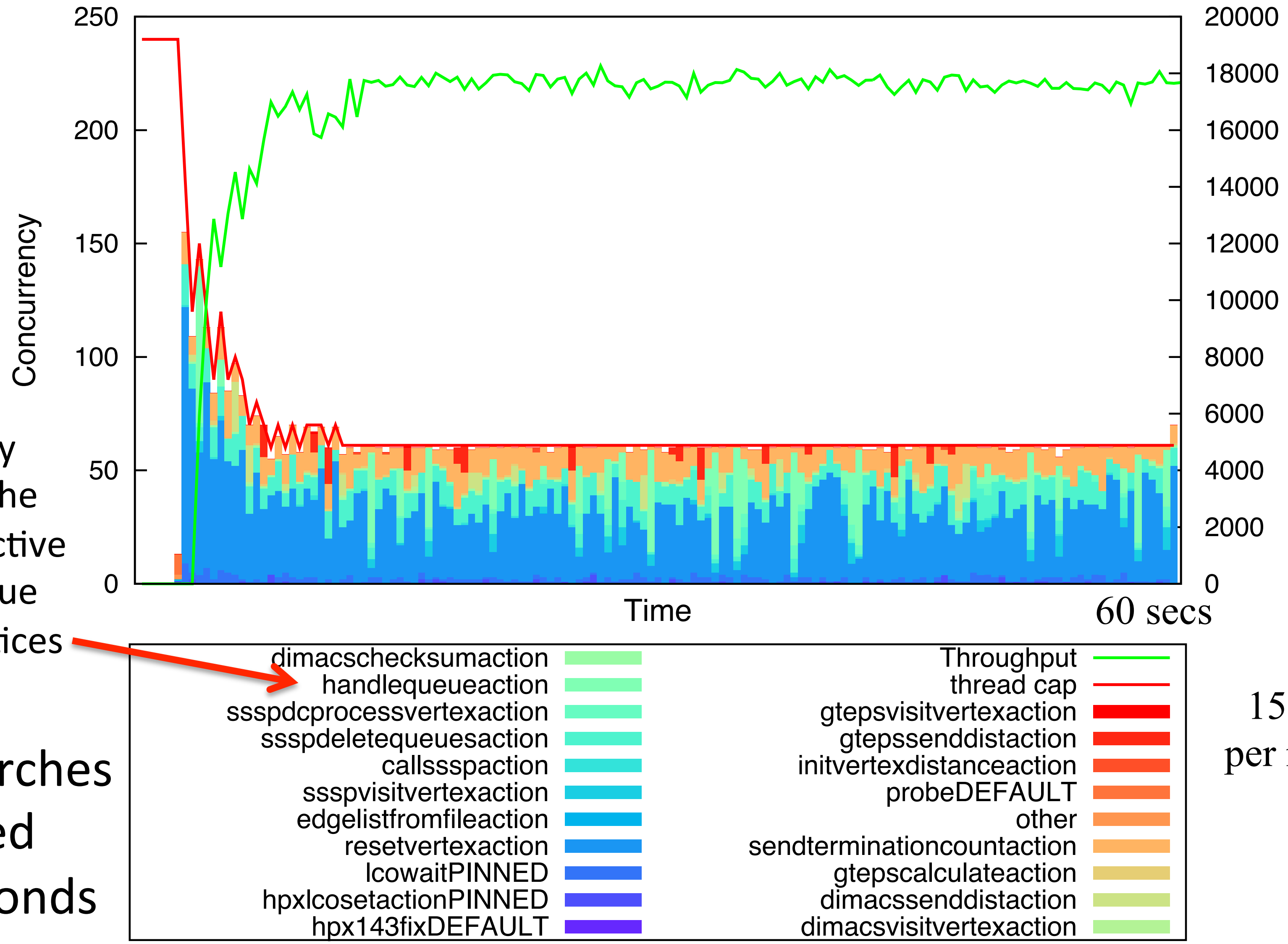


Only 61 threads (6 on 9 nodes, 7 on 1 nodes) are busy

150W per node

6929 searches performed In 60 seconds

SSSP with Throughput Policy



Used in policy to optimize the number of active threads (queue contains vertices to explore)

6929 searches performed In 60 seconds

150W per node

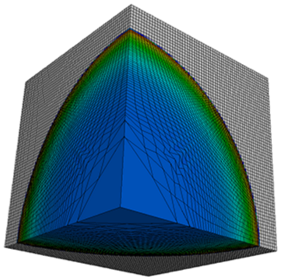
SSSP Performance Explanation

- Less threads = less contention for task yield locks (network)
- Tasks yield when waiting on network (network bound)
- Threads contend waiting on remote actions

TEPS (Traversed Edges Per Second)

Metric	Baseline	With APEX	Difference
Searches Done	1962	6929	**353.16%**
Cycles	6.91756E+12	2.81473E+12	**40.69%**
Instructions	3.17485E+12	2.01608E+12	**63.50%**
L2 Cache Misses	7986640437	8570692088	**107.31%**
IPC	0.458955	0.716263	**156.06%**
INS/L2TCM	397.52	235.23	**59.17%**
min_TEPS	7.23E+04	9.47E+04	**130.87%**
median TEPS	1.36E+05	4.95E+05	**365.01%**
max_TEPS	2.51E+05	7.63E+05	**303.36%**

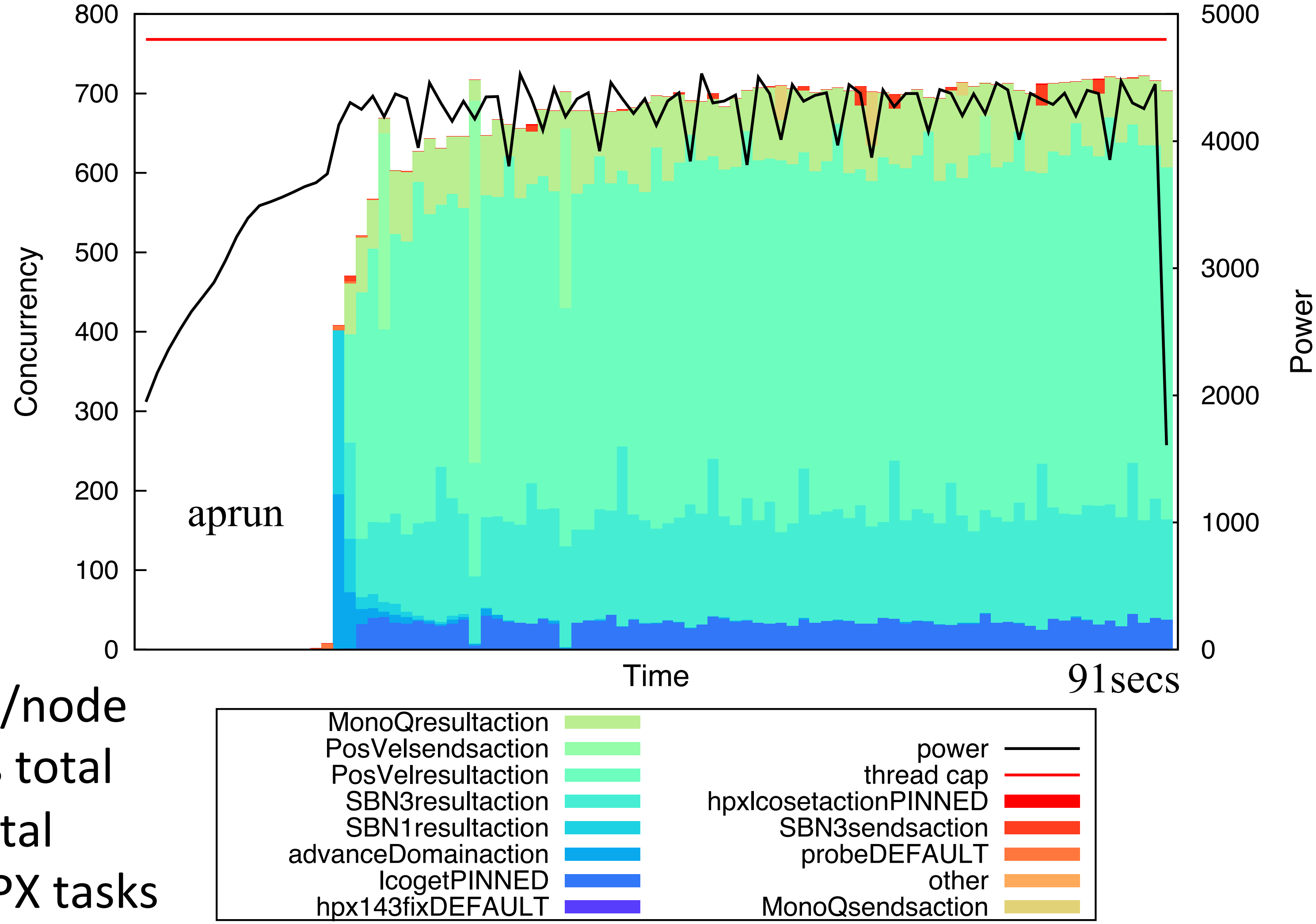
Throttling for Power



- Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH)
  - Proxy application in DOE co-design efforts for exascale
  - CPU bounded in most implementations (use HPX-5)
- Develop an APEX policy for power
  - Threads are idled in HPX to keep node under power cap
  - Use hysteresis of last 3 observations
  - If Power < low cap ➡ increase thread cap
  - If Power > high cap ➡ decrease thread cap
- HPX thread scheduler modified to idle/activate threads per cap
- Test example:
  - 343 domains, nx = 48, 100 iterations
  - 16 nodes of Edison, Cray XC30
  - Baseline vs. Throttled (200W per node high power cap)

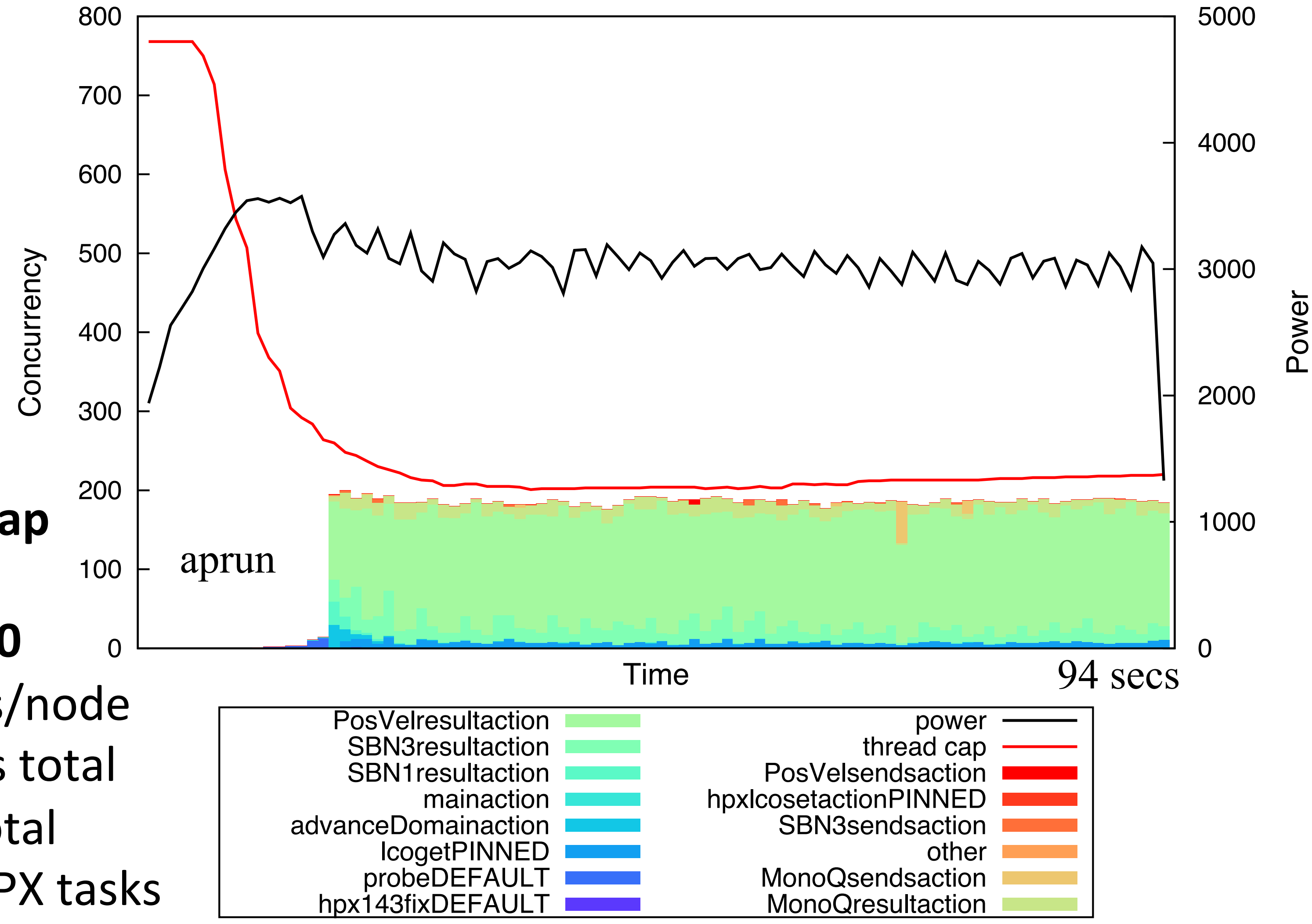
LULESH image, source: Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Technical Report, LLNL-TR-490254

LULESH Baseline



No power cap
No thread cap
768 threads
Avg. 247 Watts/node
~360 kiloJoules total
~91 seconds total
~74 seconds HPX tasks

LULESH Throttled by Power Cap



200W power cap
768 threads
throttled to **220**
Avg. 186 Watts/node
~280 kiloJoules total
~94 seconds total
~77 seconds HPX tasks

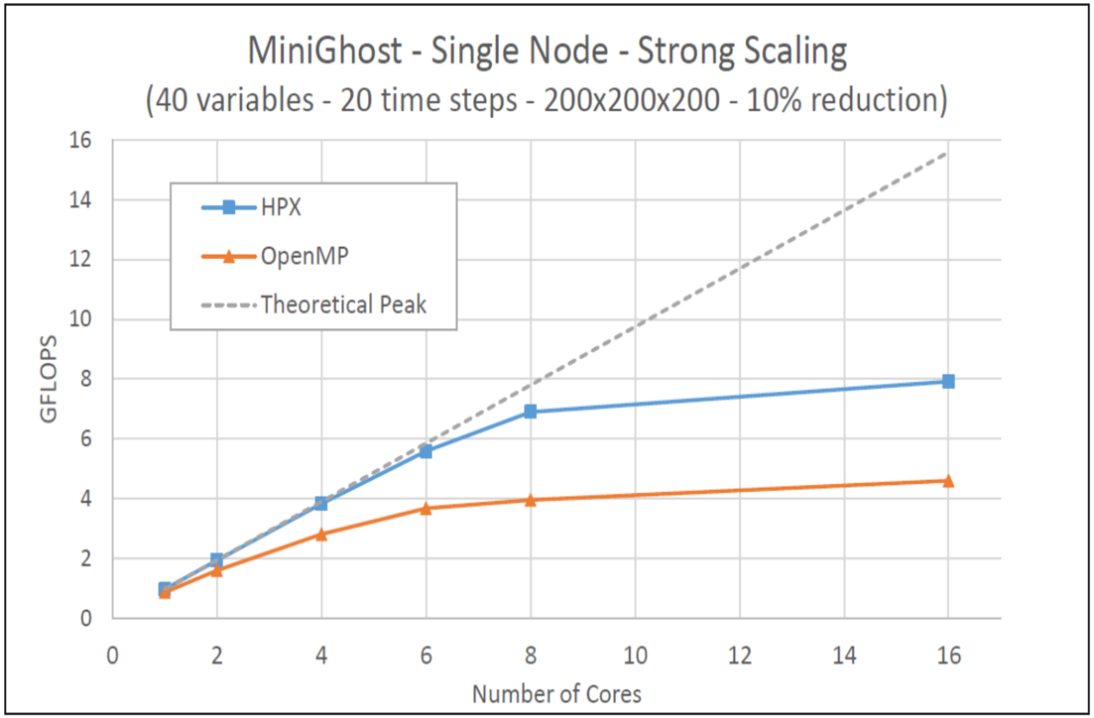
LULESH Performance Explanation

- 768 vs. 220 threads (after throttling)
- 360 kJ vs. 280 kJ total energy consumed (~22% decrease)
- 75.24 vs. 77.96 seconds in HPX (~3.6% increase)
- Big reduction in yielded action stalls (in thread scheduler)
  – Less contention for network access
- Hypothesis: LULESH implementation is showing signs of being network bound - spatial locality of subdomains is not maintained during decomposition

Metric	Baseline	Throttled	% Difference
Cycles	1.11341E+13	3.88187E+12	**34.865%**
Instructions	7.33378E+12	5.37177E+12	**73.247%**
L2 Cache Misses	8422448397	3894908172	**46.244%**
IPC	0.658677	1.38381	**210.089%**
INS/L2CM	870.742	1379.18	**158.391%**

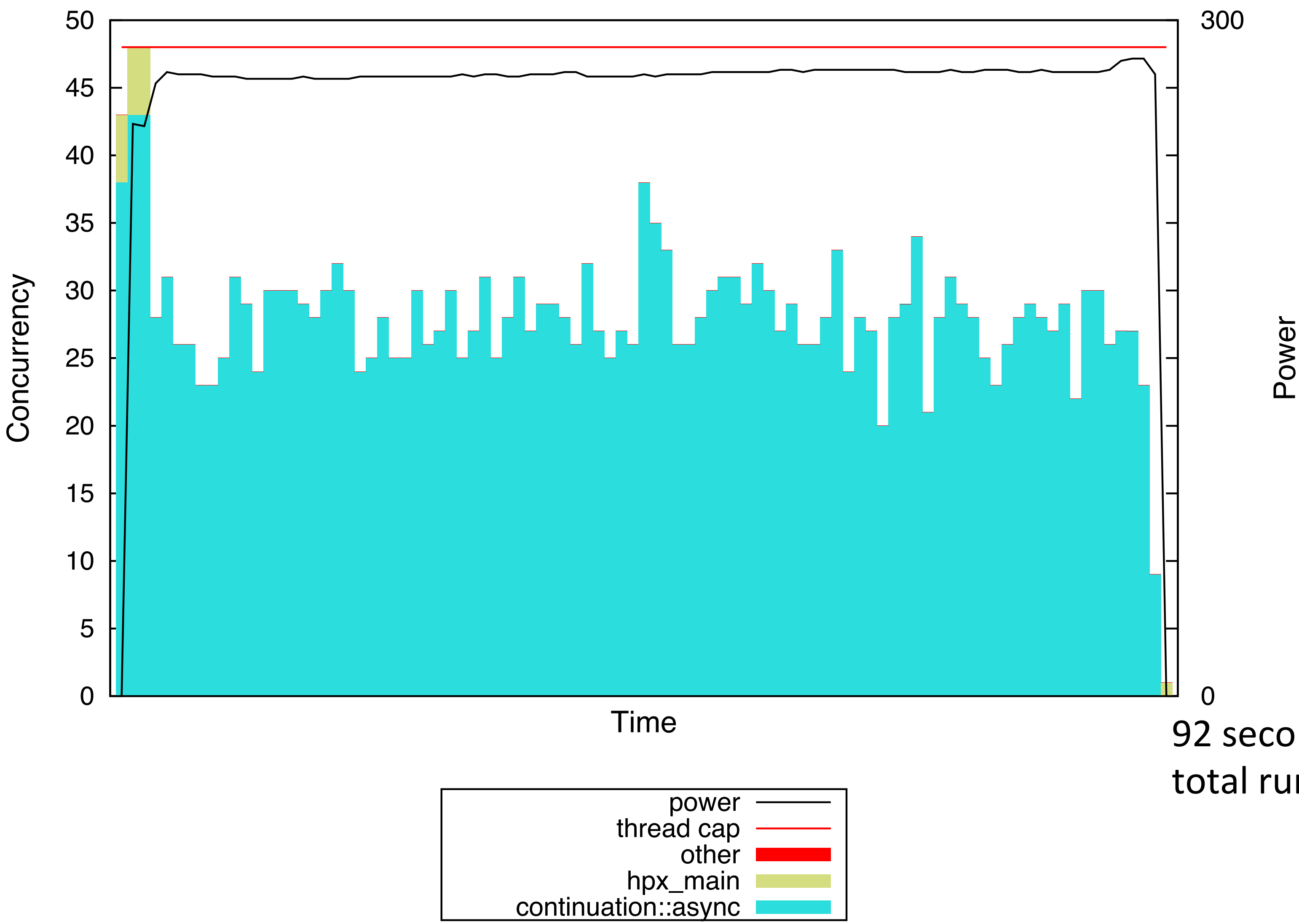
Throttling for Power

- MiniGhost
  - Mantevo finite different miniapp
  - Implements a difference stencil across homogenous 3D domain
  - Ported to HPX-3 from OpenMP+MPI
  - HPX-3 has better performance than OpenMP version



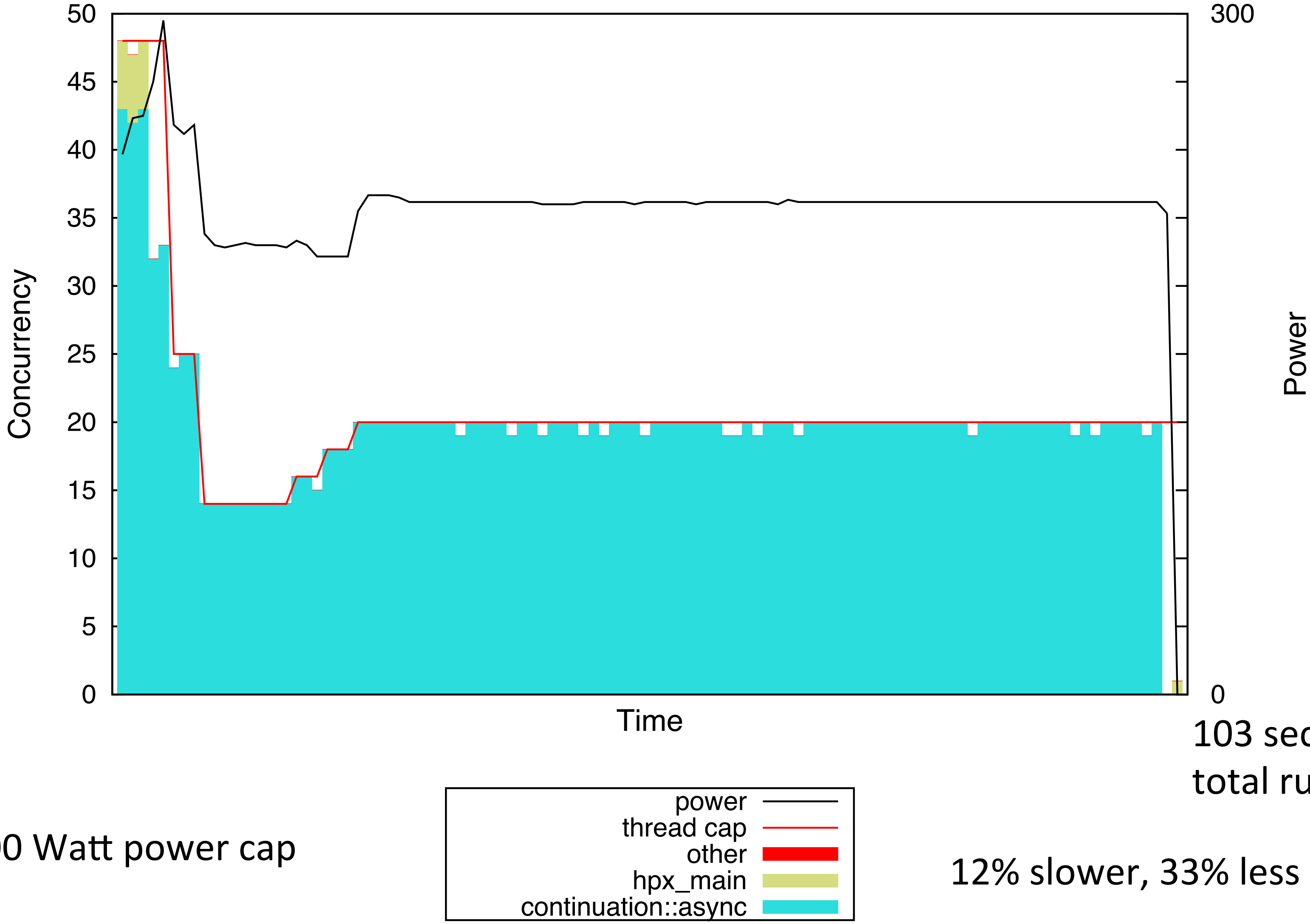
- Diminishing returns with added cores per node
  - Can throttle for energy without substantial performance impact

MiniGhost Baseline



92 seconds total runtime

MiniGhost Throttled



103 seconds total runtime

200 Watt power cap

12% slower, 33% less power

Future Work, Discussion

- Conduct more robust experiments and at larger scales on different platforms
- More, better policy rules
  - Runtime and operating system
  - Application and device-specific (*in progress)
  - Global policies (*in progress)
- Multi-objective optimization
- Integration into HPX-5, HPX-3 code bases
- API refinements for general purpose usage
- Global data exchange – who does it, and when?
- "MPMD" processing – how to (whether to?) sandbox APEX
- Source code: https://github.com/khuck/xpress-apex

Acknowledgements

- Support for this work was provided through Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (and Basic Energy Sciences/Biological and Environmental Research/High Energy Physics/Fusion Energy Sciences/Nuclear Physics) under award numbers DE-SC0008638, DE-SC0008704, DE- FG02-11ER26050 and DE-SC0006925.
- Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. DOE's National Nuclear Security Administration under contract DE- AC04-94AL85000.
- This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Sandia National Laboratories

SciDAC Scientific Discovery through Advanced Computing

NERSC