Application Heartbeats

Henry Hoffmann, Jonathan Eastep, Marco Santambrogio,
Jason Miller, Anant Agarwal

CSAIL
Massachusetts Institute of Technology
Cambridge, MA 02139

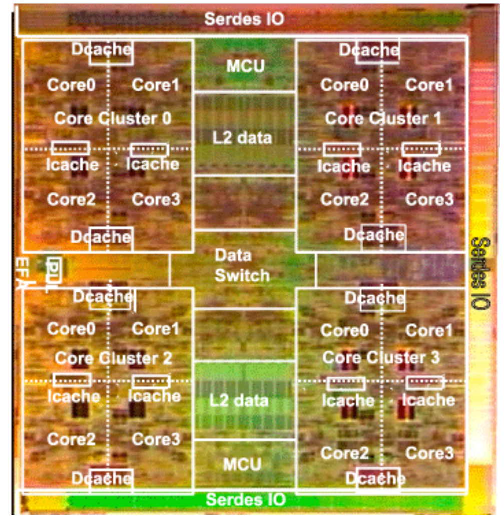
http://groups.csail.mit.edu/carbon/heartbeats

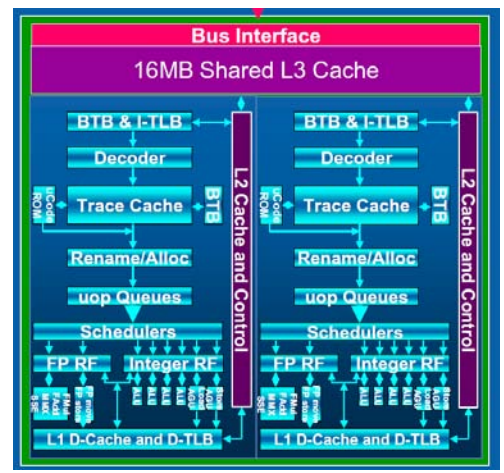
Outline

- Introduction/Motivation
  - Problem: Monitoring applications in self-tuning systems
  - Solution: Standard interface expresses performance/goals
- Application Heartbeats
- Experiments
- Conclusion

As System Complexity Increases, Self-Tuning Systems Emerge

- System Complexity is Skyrocketing
  - Multicore processors
  - Parallel communication libraries
  - Heterogeneous architectures
  - Distributed, deep memory hierarchies
  - Special-purpose functional units
  - Unreliable components
  - New constraints: power, energy, wire delay
- Application programmers must be experts in systems and apps



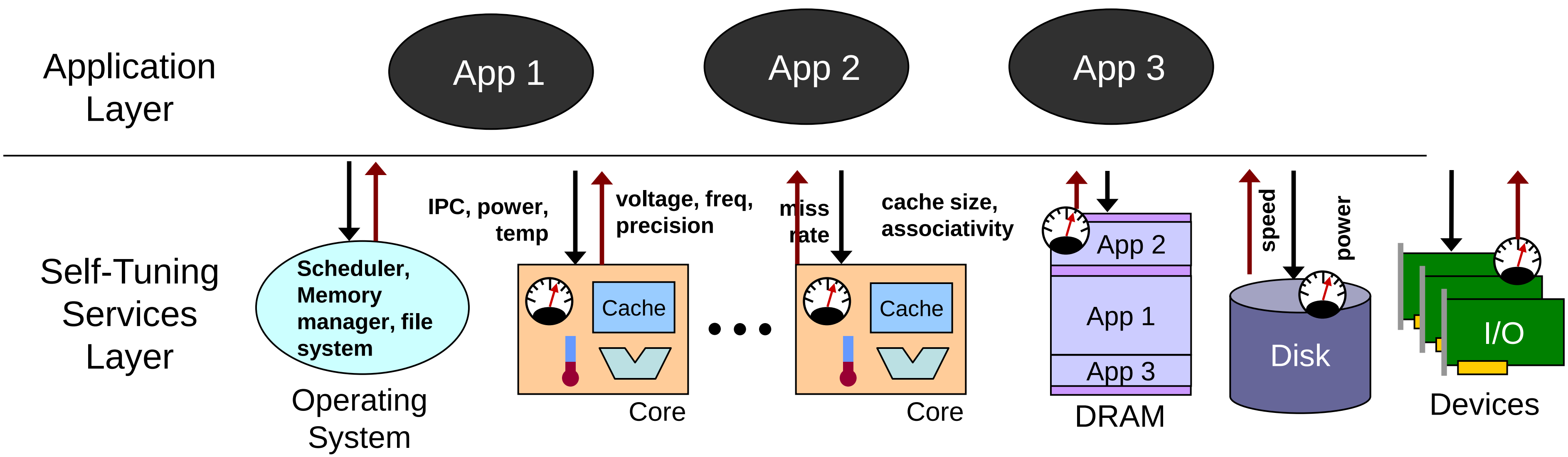


Possible Solution: Self-Tuning Systems

Systems observe their runtime behavior, learn, and take actions to meet desired goals

Self-tuning Systems Must Monitor the Applications They Support

Currently, applications run as performance black-boxes:

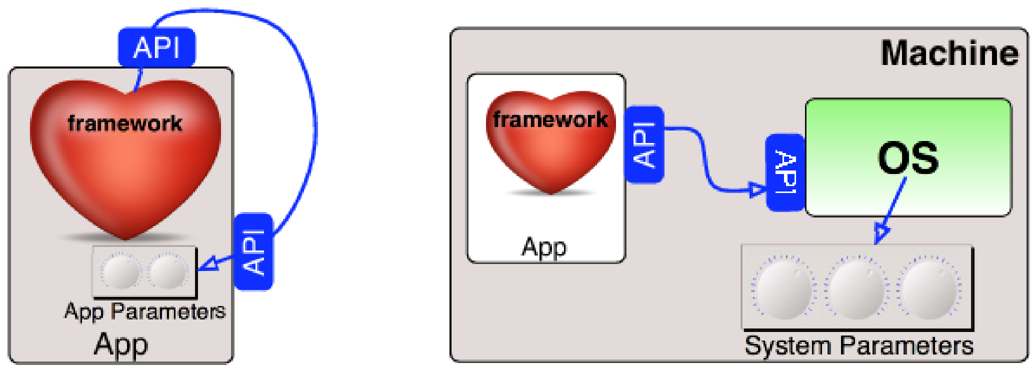


We propose Application Heartbeats as a standard API for applications to specify their goals and performance to self-tuning system services

Outline

- Introduction/Motivation
- Application Heartbeats
  - Idea
  - Interface
- Experiments
- Conclusion

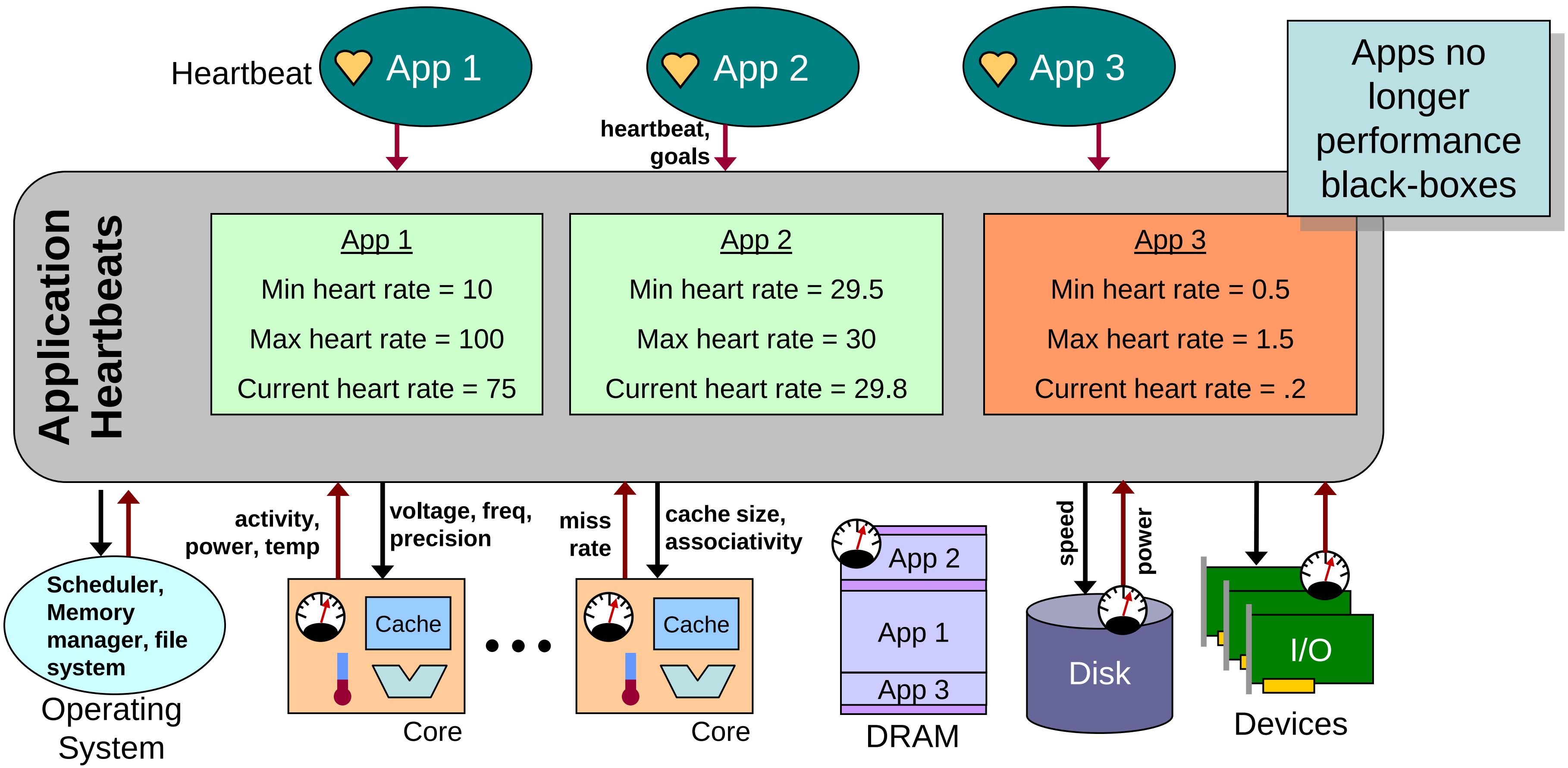
The Application Heartbeats Idea



(a) (b)

- At key intervals, apps issue a heartbeat using a simple function call
- Apps also register desired performance with other function calls
- The performance (heart rate) can be read within the application (a) or by another process (b)
- If performance is low the system adapts to increase performance

Application Heartbeats Provide Standard API for Expressing Performance & Goals



- Application Heartbeats express goals and current performance
- System software can use Heartbeats to directly measure performance

Heartbeat API Functions

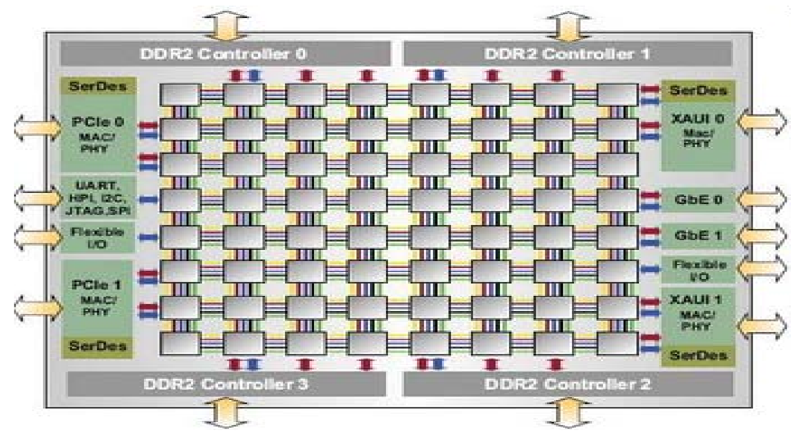
Function	Parameters	Description
heartbeat_initialize	[int] window_size	Initialize the heartbeat object to collect heartbeats. Uses a sliding window of window_size to calculate current hear trate
heartbeat	[int] tag	Records a heartbeat with a given tag
hb_get_current_rate		Returns the current heart rate averaged over the last window_size heartbeats
hb_set_target_rate	[float] min, [float] max	Sets the desired min and max heart rates for this app
hb_get_target_min_rate		Returns the minimum desired heart rate
hb_get_target_max_rate		Returns the maximum desired heart rate
hb_set_target_latency	[float] min, [float] max, [int] tag1, [int] tag2	Sets the desired latency between heartbeats with tags tag1 and tag2
hb_get_min_latency	[int] tag1, [int] tag2	Returns the minimum desired latency between two tags
hb_get_max_latency	[int] tag1, [int] tag2	Returns the maximum desired latency between two tags
hb_get_history	[int] n	Returns all heartbeat information for the last n heartbeats

Heartbeat API allows direct communication of performance and goals

Heartbeats Reference Implementations

http://groups.csail.mit.edu/carbon/heartbeats

Callable from C/C++



- Files for distributed computing
- Performance[1]
  - Throughput: ~0.900 Kbeat/s
  - Latency: ~1000 μs

- Shared Memory for multicore
- Performance[2]
  - Throughput: ~1500 Kbeat/s
  - Latency: ~1.5 μs

1. Intel Xeon servers @3.16 GHz with :Linux NFS
2. Intel Xeon servers @ 3.16 GHz with Linux and POSIX shared memory

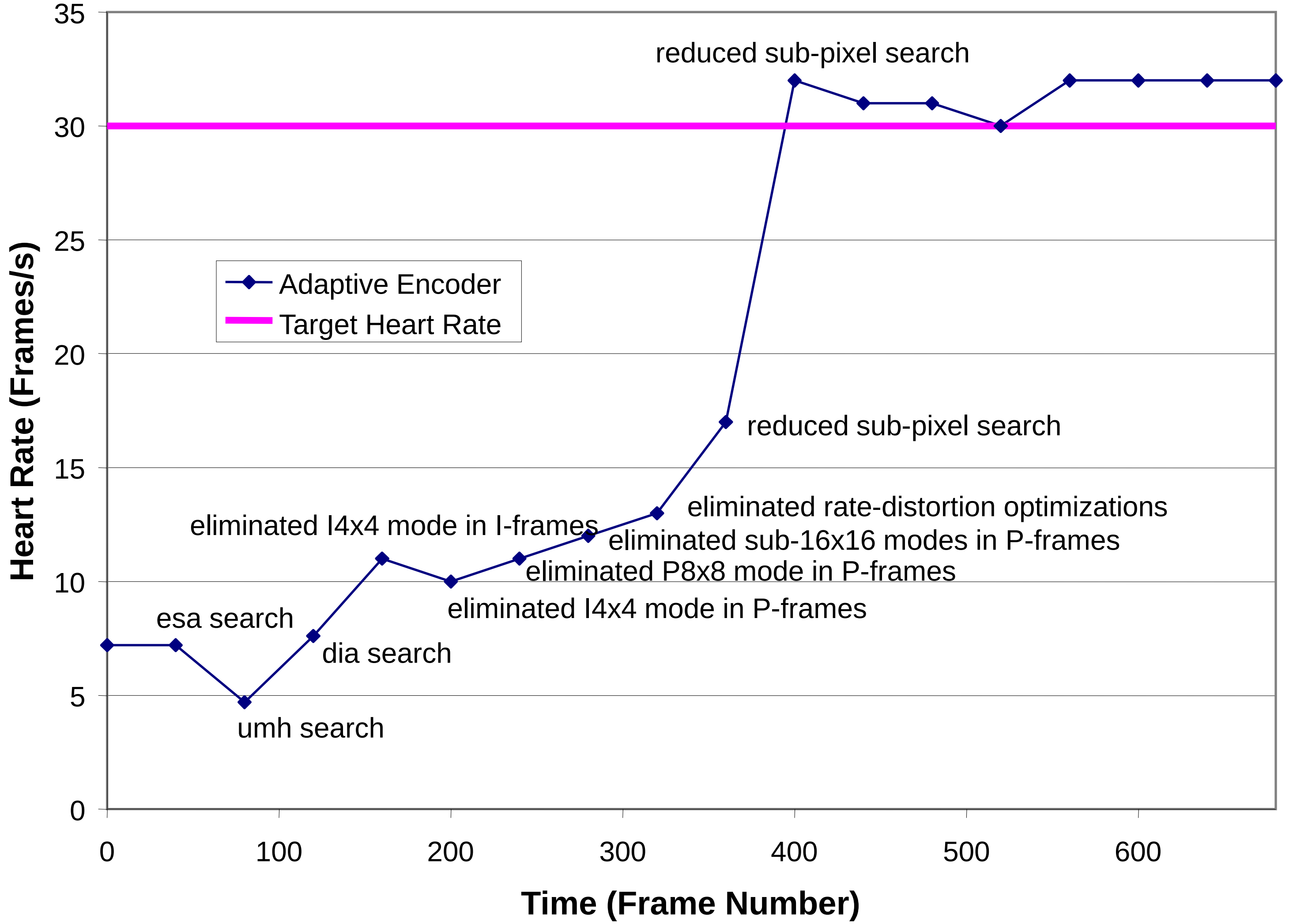
Outline

- Introduction/Motivation
- Application Heartbeats
- Experiments
  - Heartbeat use within an application
  - Heartbeat use by an external system
  - Other systems using Heartbeats
- Conclusion

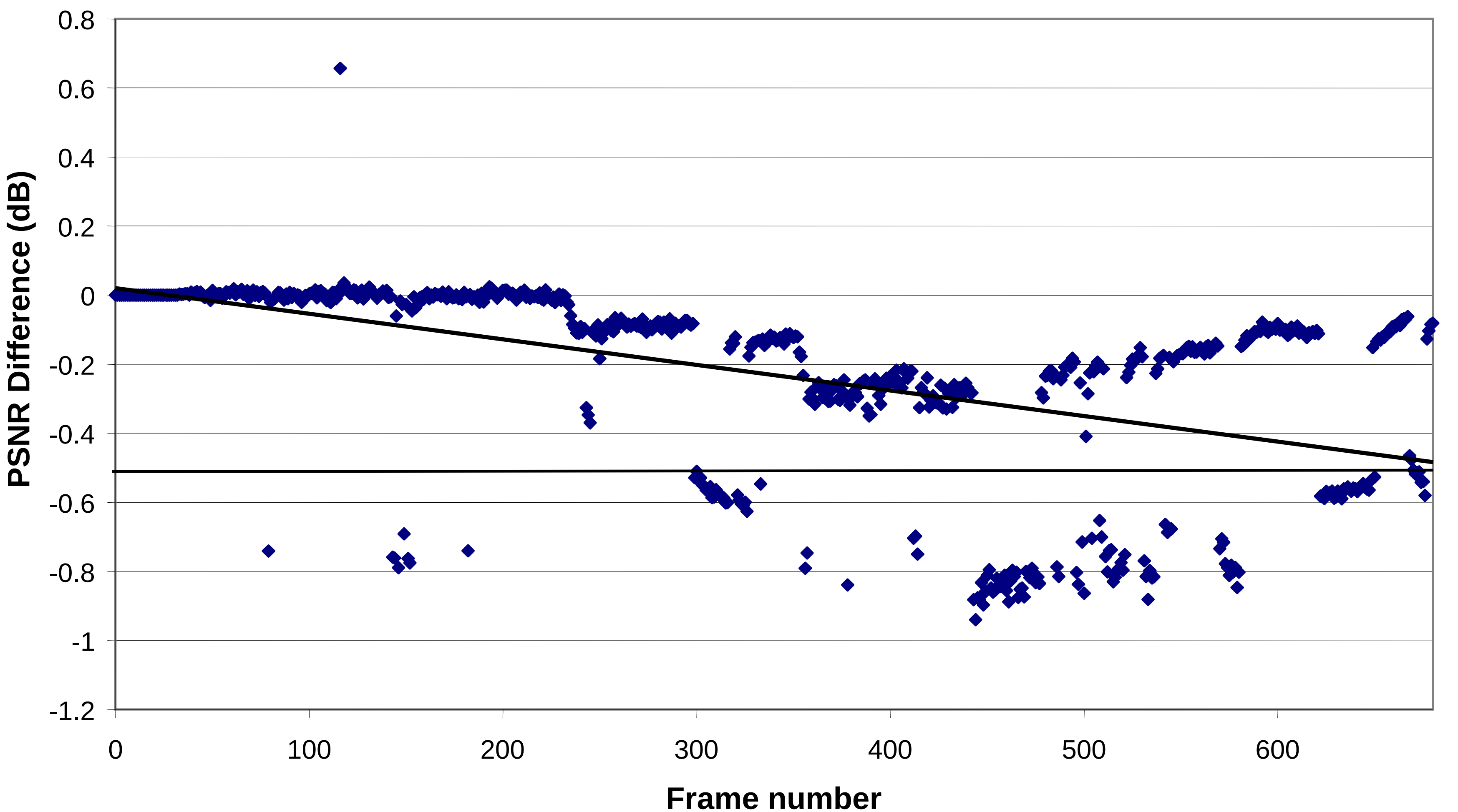
Experiment 1: Internal Heartbeat Usage

- Experiment 1: Adaptive H.264 Encoder
- Goal: produce the highest quality video in real-time
- Method:
  - A heartbeat is registered for each frame (frame rate = heart rate)
  - Encoder reads heartbeat and changes algorithm to reach target
- Results:
  - Now the encoder is fast and still high quality
  - Achieve target performance with barely visible quality loss

Example 1: Performance



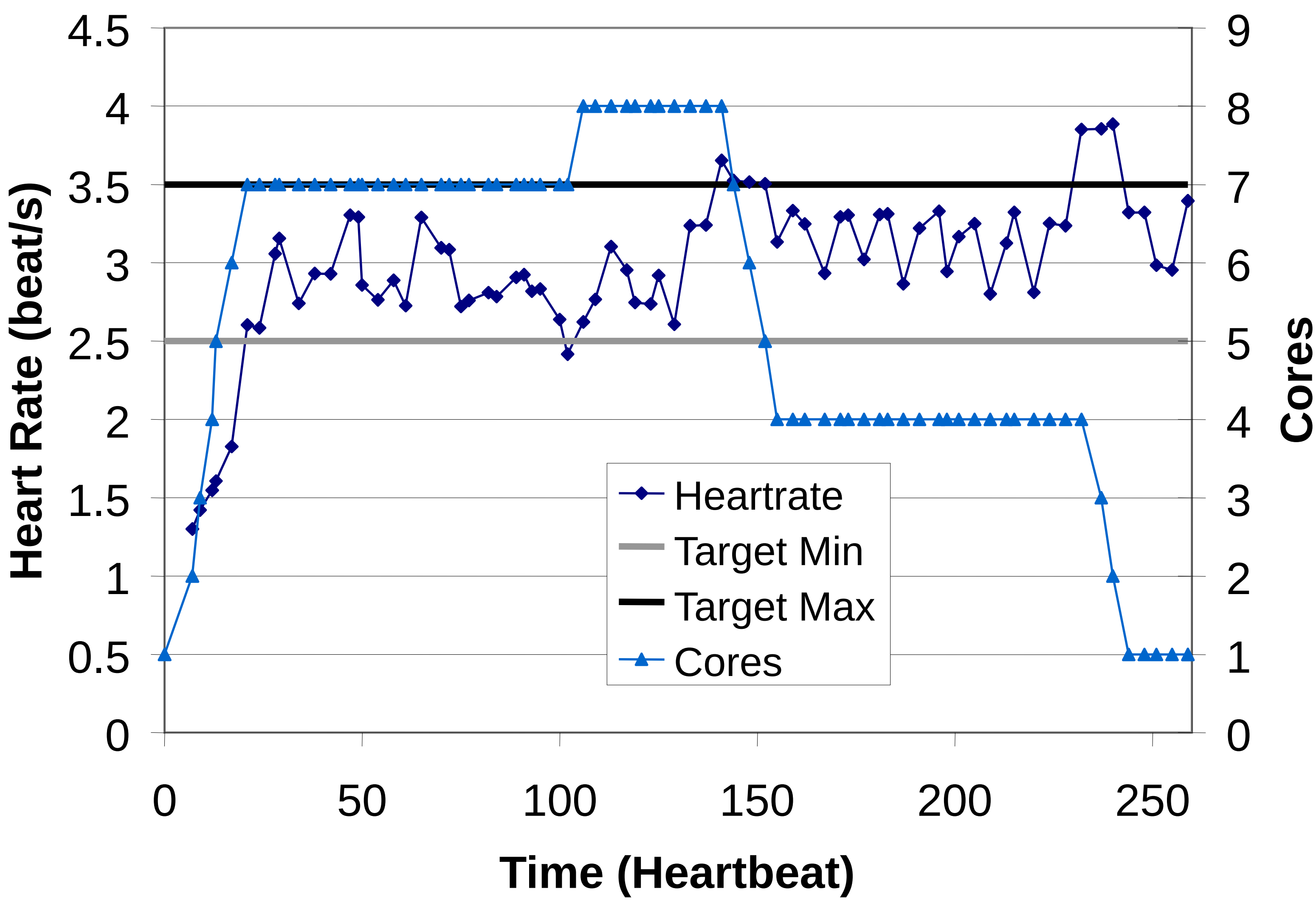
Example 1: Image Quality



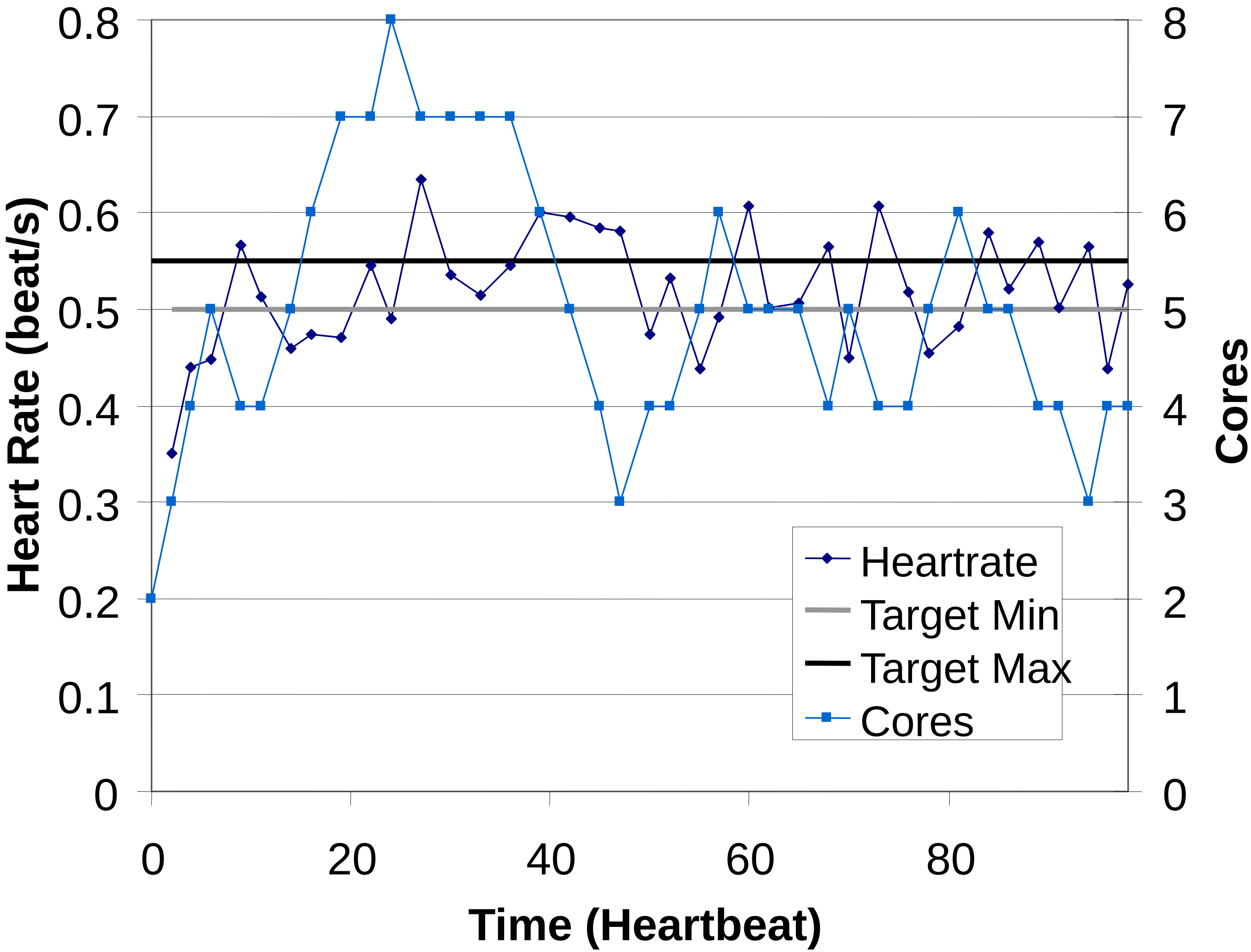
Example 2: External Heartbeat Usage

- Experiment 2: External System Reads Heart Rate and Assigns Cores
- Goal: Assign cores to keep performance within target range
- Method: Use PARSEC benchmarks
  - Target heart rates set to be achievable using less than full number of cores
- Results:
  - The scheduler keeps the applications running at the target speed
  - Scheduler can adapt to changes in the difficulty of the inputs

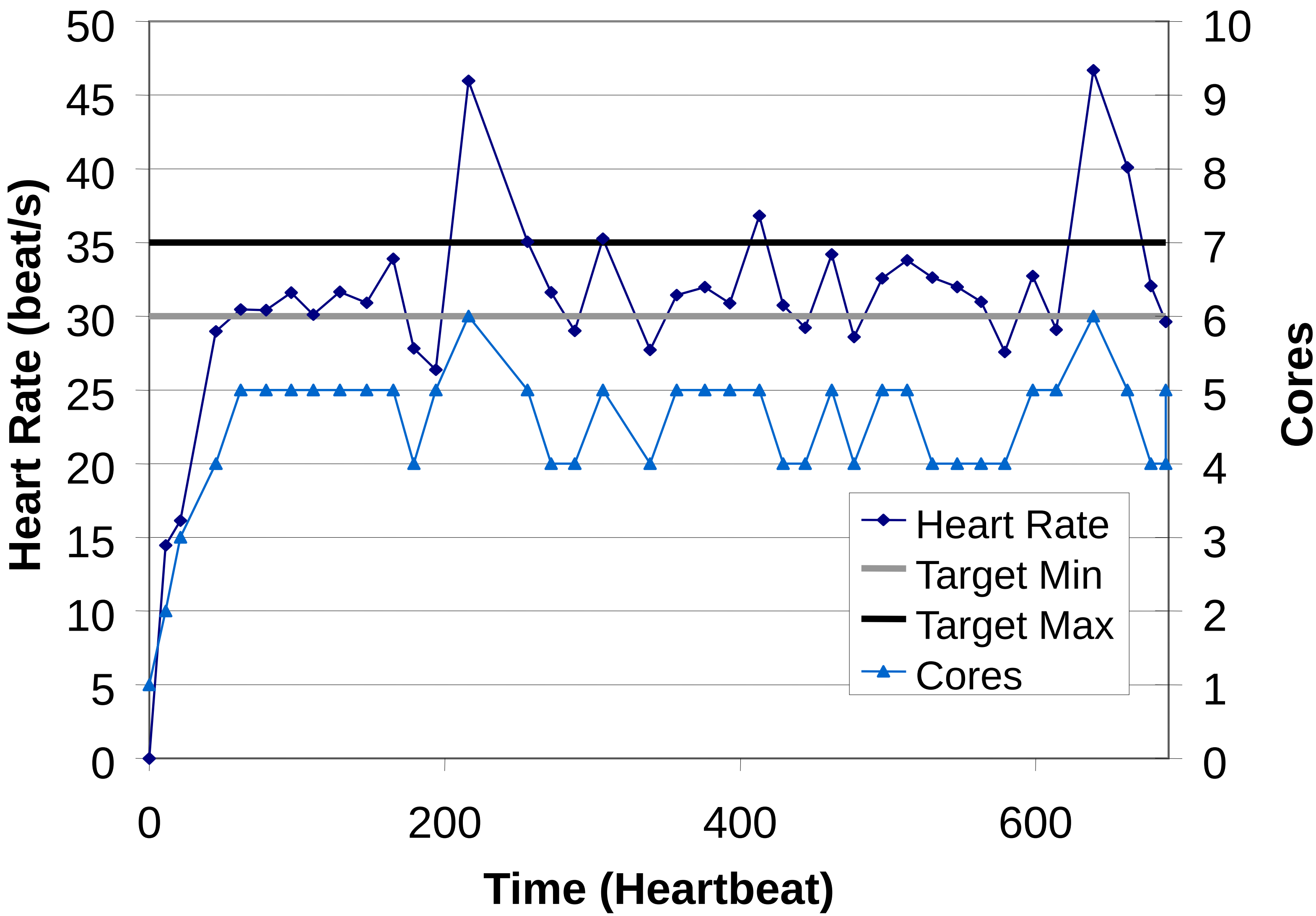
Example 2: bodytrack



Example 2: streammcluster



Example 2: x264



Other Heartbeat Uses

- SpeedPress compiler and SpeedGuard runtime system
  - The SpeedPress compiler discovers possible quality-of-service/ performance tradeoffs
    - Achieve up to 2x speedup for 5% QoS loss
  - The SpeedGuard runtime makes these tradeoffs dynamically in response to maintain a given heart rate in the face of environmental changes

More detail available in:

Hoffmann, Misailovic, Sidiroglou, Agarwal, Rinard. Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures. MIT-CSAIL-TR-2209-042. August, 2009.

- SmartLocks
  - Subject of an upcoming SMART talk

Outline

- Introduction/Motivation
- Application Heartbeats
- Experiments
- Conclusion
  - Request for feedback/usage
  - Summary

Request for Feedback

- Thanks to the reviewers for their feedback, but we need more…

- Heartbeat code is available online
  `http://groups.csail.mit.edu/carbon/heartbeats`

- We need your feedback!
  - If you have an self-tuning system service that could benefit from being able to directly measure an application's performance try the interface
  - Let us know what you think

Summary

- Presented the Application Heartbeat interface
  - API provides a standard means for an application to make its performance and goals known
- Presented several experiments showing basic usage
  - Several other systems at MIT are using Heartbeats in more advanced applications
- Requested feedback from the community

Adaptive Scheduling Algorithm

- Take average heart rate over last 20 beats
- If heartbeat < target min
  - Add a core
  - Wait for 20 beats and reapeat
- Else if heartbeat > target max
  - Remove a core
  - Wait for 20 beats and repeat
- Else
  - Repeat