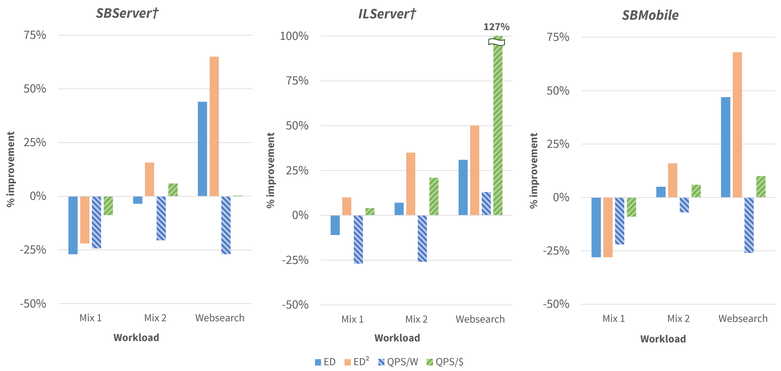
Dynamic Management of TurboMode in Modern Multi-Core Chips

David Lo and Christos Kozyrakis

What is TurboMode (TM)?

- **Enablers**
  - TDP per core is less than TDP for entire multi-core chip
  - TDP is for worst case, average case power can be less
- Overclock cores when there is extra thermal headroom
- **Significant**: up to 59% overclock
- **Ubiquitous:** present in all modern x86 multi-cores
- **Problematic**: TM is adjusted by hardware, software can only turn it on or off

Motivation



Metric change when TM is always on vs. when TM is always off

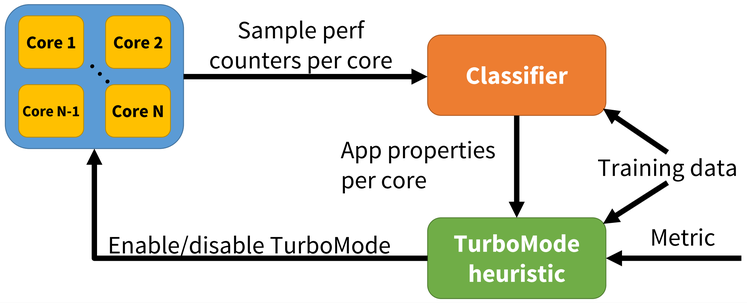
Some terminology:

- ED = Energy Delay Product
- ED^2 = Energy Delay Squared Product
- QPS/W = Throughput per Watt (OpEX)
- QPS/$ = Throughput per Dollar (CapEX)

Need a dynamic approach to determine when TM is good

autotuner implementation

- Use perf counters to predict which workloads benefit from TM on a per-metric, per-machine basis
  - **No changes to existing infrastructure needed**
  - Single active core case
  - Multiple active cores case: **interference from many sources**
  - Use machine learning to automate parts of this process
- Every 5 sec, sample perf counters and enable/disable TM



Methodology

- **Hardware**
  - SBServer: 3.20GHz->3.80GHz Core i7, single socket
  - ILServer: 1.90GHz->3.00GHz Opteron, single socket
  - SBMobile: 2.50GHz->3.60GHz Core i7 mobile, single socket
- **Benchmarks**
  - SPECCPU2006 (single app)
  - SPECCPU2006 mixes (multi-programmed)
  - SPECpower_ssj2008 (multi-threaded)
  - websearch (multithreaded)
- **Measure full system power/energy**
- **Energy proportional estimation († suffix)**
  - For SBServer and ILServer, subtract out idle system power for energy proportional estimate

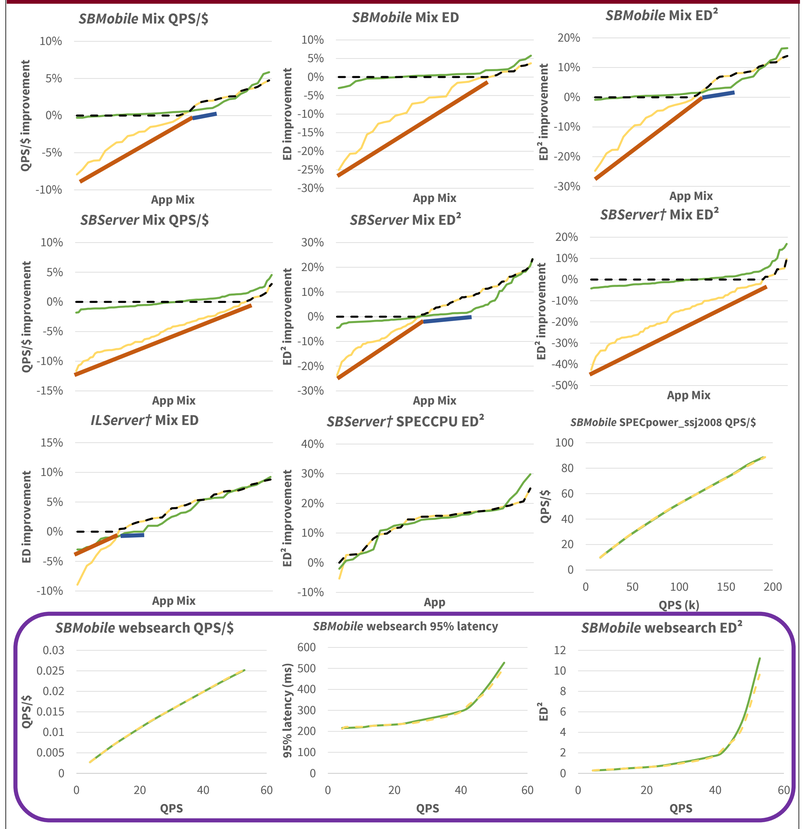
Model building

- Single active core
  - One performance counter: % cycles w/ mem. requests
  - Classifier accuracy > 96%
- Multiple active cores
  - Look for sources of interference: thermal, memory, etc.
  - For evaluated hardware and benchmarks, found that only memory interference matters
    - This could change for future generation CPUs, but technique still applies
  - Build two classifiers: *MemSensitive* and *MemInterfering*

	Machine	Classifier	Counters Used
MemSensitive	**SBServer**	Log. Reg.	% cycles w/ outstanding mem. req. % cycles front end idle
	ILServer	Naïve Bayes	L2-load-misses / instr.
	SBMobile	Naïve Bayes	% cycles w/ outstanding mem. req. % cycles front end idle
MemInterfering	**SBServer**	Log. Reg.	% cycles w/ outstanding mem. req.
	ILServer	Naïve Bayes	L3-misses # requests to mem. / instr.
	SBMobile	Log. Reg.	% cycles w/ outstanding mem. req.

  - Destructive interference predicted when two separate cores hosts a *MemSensitive* and a *MemInterfering* workload
  - Train heuristic to maximum number of interfering workloads before a metric is degraded

Evaluation



- *autotuner* prevents cases where TM degrades metrics
- Conservative heuristic misses some opportunities for TM when not too much interference exists
- Overheads of *autotuner* is small enough such that even latency critical apps (websearch) is unperturbed

Conclusions

- Use of TM can lead to big wins
  - Conversely, use of TM can also lead to large degradations
- Dynamic management of TM needed
  - Existing firmware controller insufficient
  - No “one size fits all” setting for TM across workloads and metrics
- *autotuner* can accurately prevent TM from degrading metrics while preserving benefits of TM
  - Uses machine learning on performance counters to predict optimal TM setting
  - Lightweight, can be easily deployed across entire datacenter without additional hardware