Computational Sprinting on a Hardware/Software Testbed

Arun Raghavan* Laurel Emurian* Lei Shao‡
Marios Papaefthymiou† Kevin P. Pipe†‡ Thomas F. Wenisch† Milo M. K. Martin*

* Dept. of Computer and Information Science, University of Pennsylvania
† Dept. of Electrical Engineering and Computer Science, University of Michigan
‡ Dept. of Mechanical Engineering, University of Michigan

Background

- Inflection point for CMOS scaling due to thermal constraints causing the advent of dark silicon
- Problem is particularly acute in mobile hand-held devices e.g.: Apple A5 chip
- Dark silicon can be an opportunity - modify use of chip area to deliver value
- Some applications consist of short bursts of intense computation followed by long periods of waiting for user input
- For these short bursts responsiveness can be improved by transiently exceeding the chip’s sustainable thermal limits do deliver a brief but intense burst of computation

What is ‘sprinting’?

- Computational Sprinting
  - Activate reserve cores (parallel sprinting)
  - Increase frequency or voltage (frequency sprinting)
- Increases power consumption to levels beyond the system’s cooling capacity as determined by its TDP
- Exploits thermal capacitance to buffer (absorb and hold) heat and then release it to the ambient after the sprint is over
- Prior work on sprinting was based on modelling and simulation
- The paper describes a concrete implementation of sprinting on a hardware/software testbed

Types of Sprinting

- Sprinting consumes much higher power than normal execution (up to 5x)
- The heat dissipated cause the chip temperature to rise in matter of seconds
- Temperature rise is gradual (over a few seconds) and not instantaneous
- Unabridged sprint
    - Completes execution within the thermal capacitance of the testbed, relatively simple to implement and optimise
- Truncated Sprints
    - Sprints that must be stopped due to thermal limits of the testbed, relatively more complicated implementation

Major contribution

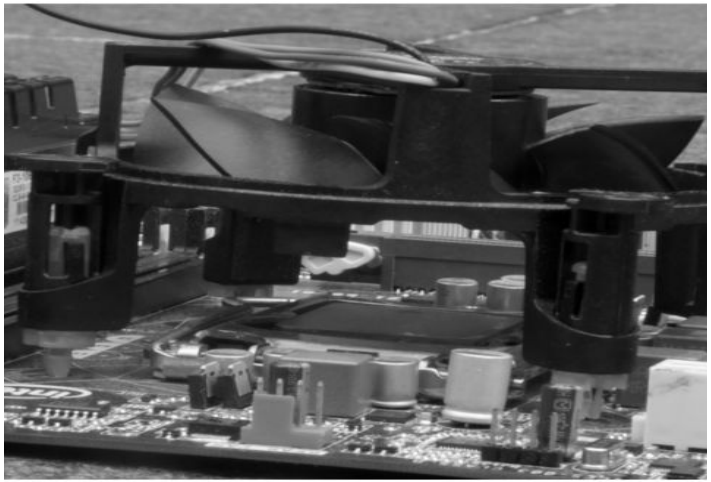
- Show that current sources of thermal capacitance are sufficient for 5x intensity sprints for up to a few seconds of duration
- Demonstrate that parallel sprinting can improve energy efficiency
- Identify potential inefficiencies of truncating sprints and propose sprint-aware task-based runtime as a remedy
- Augmentation of the testbed with a phase-change heat sink
- Demonstrate that sprint-and-rest thermally constrained (sustained) execution for long-running computations

Sprinting Testbed Hardware

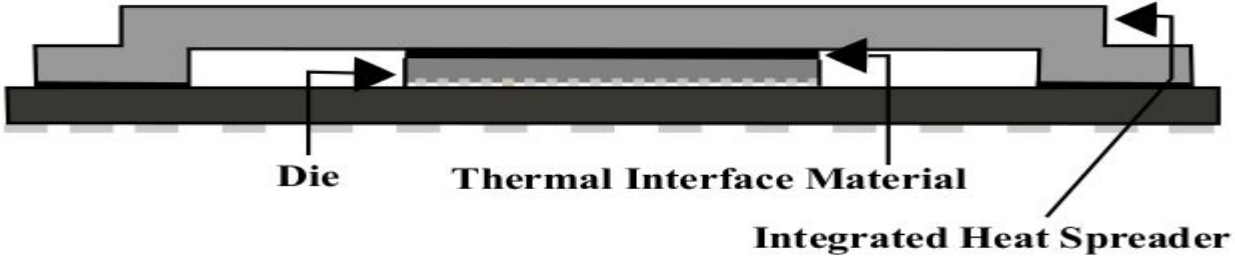
- Testbed features:
  - Intel Core i7 2600 Quad-core ‘Sandy-Bridge”
  - Small variable speed fan
  - Integrated heat spreader
  - Lowest configurable frequency = 1.6GHz
  - Highest configurable frequency = 3.2 GHz
  - Power draw of single core @ 1.6GHz = 9.5W
  - Power draw with 4 cores @ 3.2GHz = 50W (5x)
- Variable Fan
  - Sustainable TDP can be varied by varying the fan speed
  - Speed tuned such that the die temperature saturates at 75 C
  - The fan is required for indefinitely running even a single core at min frequency/voltage

Sprinting Testbed Hardware

- Integrated Heat Spreader
  - Testbed has no heat sink (like a mobile system)
  - It has a copper IHS that spreads heat
    - On the die to reduce hotspot)
    - From the die to the top of the package (to facilitate cooling)
  - Based on copper's chemical characteristics, an idle temperature of 50 C and a max. Temperature of 75 C during sprinting, the IHS can store 188J of heat
  - This leads us to a theoretical limit on the duration of the sprint = 4.7s at 40W above the sustainable TDP(50W in total)
  - But head spreading is not instantaneous, thus the actual limit is slightly lower



(a) Testbed setup



(b) Package internals

Unabridged Sprint

- Estimate for sprint duration = 4.7s
- Actual value based on experimental trials @ 3.2Ghz with 4 cores = 3s
- Theoretical speedup benefit = 8x
  (2x freq., 4x cores)
- Total power dissipated = 50W (5x sustainable)
- Benchmarks used: workloads from image and vision processing domains
- Experiments:
  - Maximum intensity sprints with each workload
  - Comparison of power and temperature between sustainable and max. Intensity sprint for single workload (sobel)

Kernel	Description
`sobel`	Edge detection filter; parallelized with OpenMP
`disparity`	Stereo image disparity detection; adapted from [51]
`segment`	Image feature classification; adapted from [51]
`kmeans`	Partition-based clustering; parallelized with OpenMP
`feature`	Feature extraction (SURF) from [14]
`texture`	Image composition; adapted from [51]

Table 1. Parallel workloads used to evaluate sprinting.

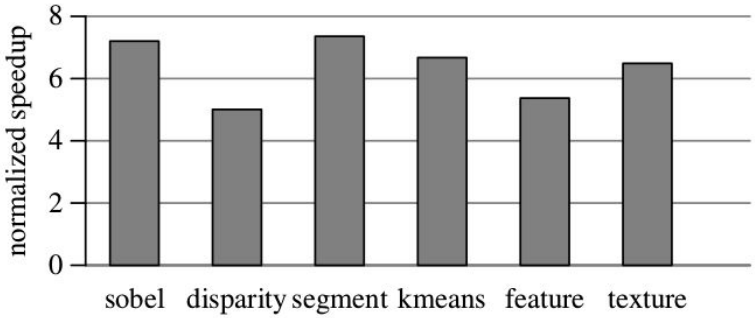
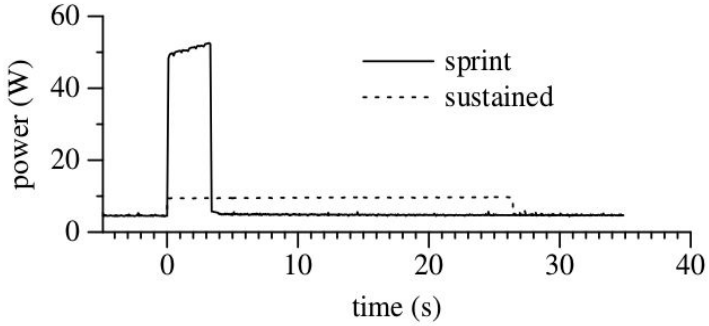
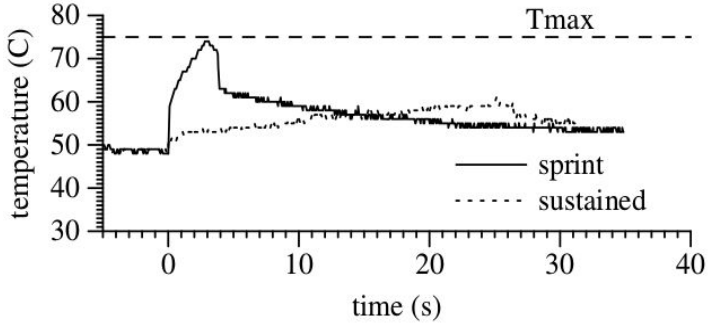


Figure 2. Speedup benefits of unabridged sprinting using four cores at 3.2 Ghz over a single-core 1.6 Ghz baseline.



(a) Power



(b) Temperature

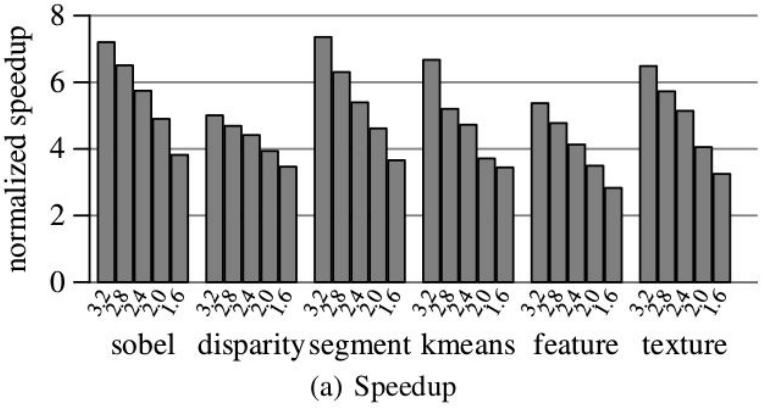
Figure 3. Chip power and thermal behavior for sustained and sprinting operation.

Analysis of Unabridged Sprinting experiments

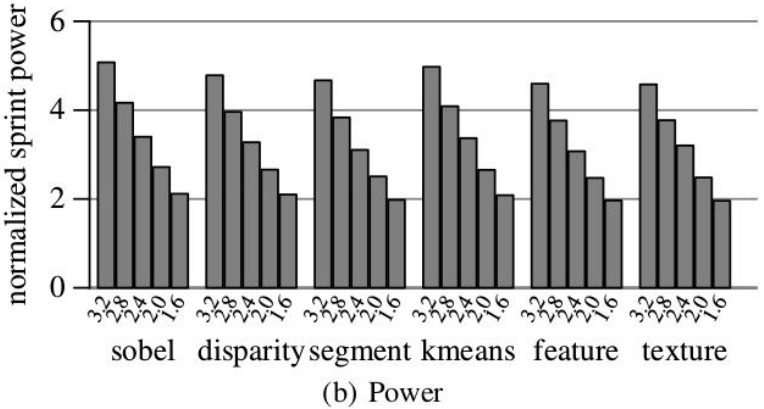
- Sprinting can result in net gains in energy efficiency by:
  - Amortizing the fixed uncore power consumption over a larger number of active cores
  - Capturing race-to-idle effects
- When both frequency and voltage are increased, a super-linear increase in power
- However, the observed increase is sub-linear
- Last level cache and ring-interconnect are always active and add to the ‘uncore’ power of the chip
- Thus when no. of cores is quadrupled, power consumption only increases power by 2x

Analysis of Unabridged Sprinting experiments

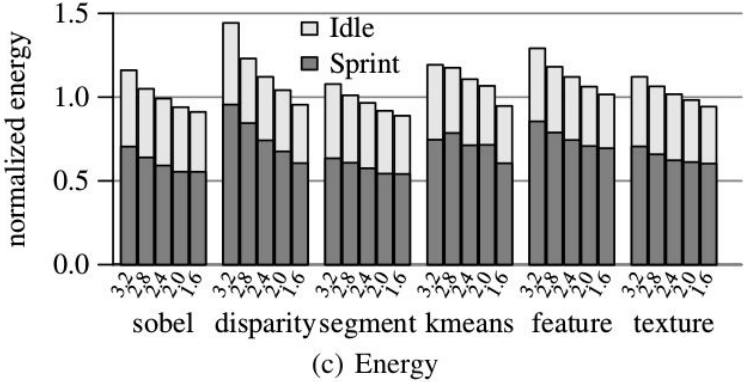
- Sprinting with lower frequency can improve energy efficiency
- Power consumption falls much more than speedup
- Energy consumption for the overall computation is less at lower clock frequencies
- Total energy = Sprint energy + idle energy
- This sum can then be compared to the much slower sustainable computation (1 core @ 1.6GHz)
- Ignoring idle power, sprinting consumes -30% at maximum frequency (up to -40% at lower frequencies)
- But with idle power, energy consumed is +21% whereas for 1.6Ghz it is -6%



(a) Speedup



(b) Power



(c) Energy

Figure 4. Speedup, power, and energy (normalized to the one-core 1.6 GHz sustainable baseline) for four cores across frequencies.

Analysis of Unabridged Sprinting experiments

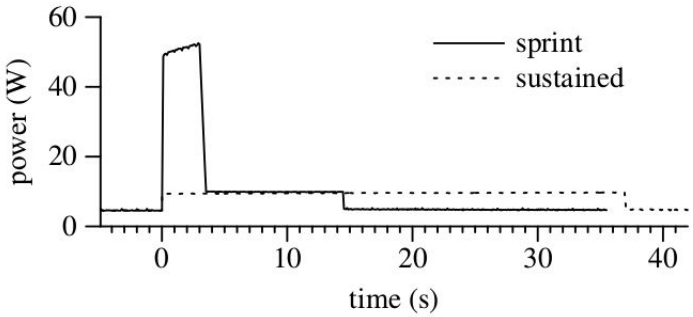
- Even on current chips that are not built with sprinting in mind, sprinting can actually reduce total energy consumption if the clock frequency is appropriately chosen
- Idle power acts as a spoiler for sprinting
- But there are encouraging signs of reduction in idle power
- Energy efficiency advantages of sprinting increase rapidly as idle power decreases
- Eg.: NVIDIA Tegra 3's vSMP/4-plus-1, ARM's big.LITTLE multicores that seek to aggressively reduce idle power

Truncated sprints

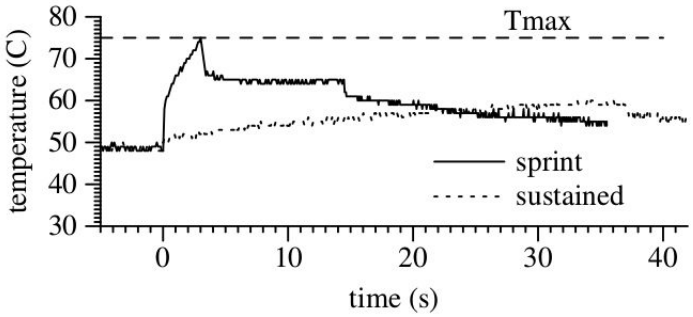
- System must avoid overheating for computation that cannot be completed entirely within one sprint duration
- Software for throttling frequency and deactivating all but one core
- For some workloads, migrating threads to a single core can cause significant overheads leading to degradation in performance and energy efficiency
- Mitigation of these overheads becomes necessary to make sprinting energy efficient vis-a-vis sustainable non-sprinting mode

Truncated Sprints - Implementing and Evaluating Truncation

- Testbed software monitors die temperature using the on-die temperature sensor
- When temperature exceeds T(max), the software truncates the sprint by
  - Pinning all threads to a single core thus forcing the OS to migrate threads
  - Disabling the now-idle cores
  - Changing frequency of the single remaining core to the lowest possible frequency (1.6GHz)
- This process is completed using system calls and the standard Linux ACPI interface
- When the temperature reaches T(max), the power falls abruptly from 55W to 9.5W
- The temperature stops rising as the system's power consumption now matches the rate of cooling dissipation



(a) Power



(b) Temperature

Figure 5. Power and thermal response for truncated sprints.

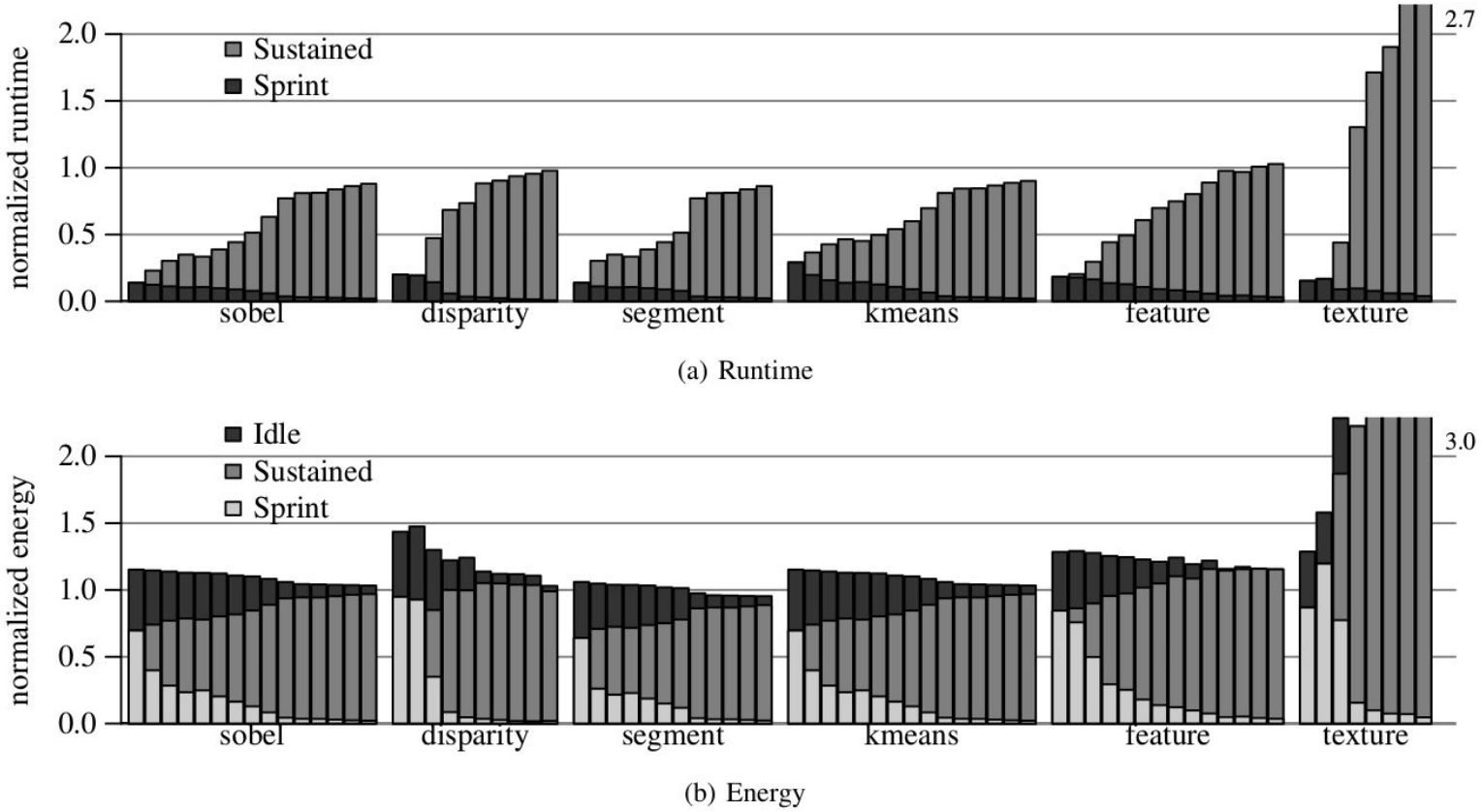


Figure 6. Runtime and energy spent during sprinting, sustained, and idle modes for 4-core sprints at 3.2 Ghz (normalized to the one-core 1.6 Ghz baseline.) Bars represent increasing computation lengths from left to right.

Truncated Sprints - Performance and Energy Penalties

- Varying computation lengths lead to different degrees of responsiveness
- As computation length increases beyond the sprint capacity, greater proportion of the time is spent computing in sustainable mode
- Thus, the gain from sprinting reduces as the length of computations increases
- Anomaly: Truncated sprinting is slower than sustainable mode for two workloads
- Degradation in responsiveness occurs due to multiplexing of all threads on a single core
- Leads to one or more of known issues such as contention on synchronization, load imbalance, convoying and frequent context switches

Truncated Sprints: Mitigating Overheads

- Penalty of oversubscription causes the ‘do no harm’ policy of sprinting to fail
- Problem particularly acute for two workloads: feature and texture
- Experiment to test such penalties:
  - Spawn N threads and run them on a single core; measure runtime

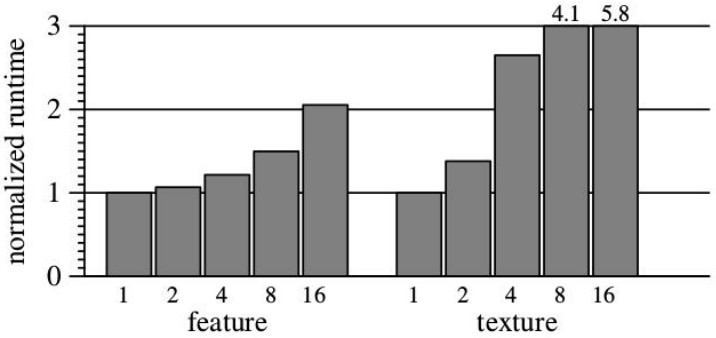


Figure 7. Runtime penalty from oversubscription with increasing numbers of threads on a single core.

Truncated Sprints: Resolving Over-subscription Penalties

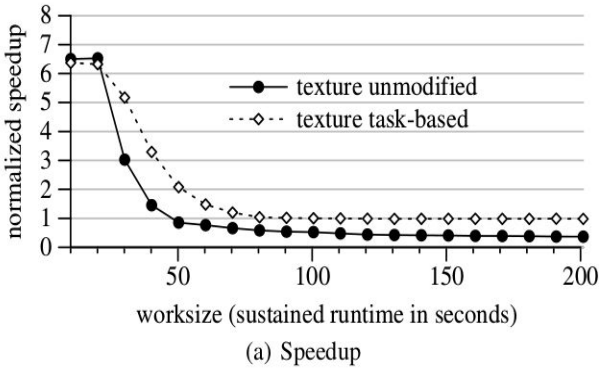
- Sprint-aware task-based parallel runtime to mitigate oversubscription penalties
- Number of software threads must be changed dynamically
- Task-queue based worker thread execution framework
- Application is divided into tasks but program is not aware of the number of worker threads
- Worker thread will first execute tasks in its local queue; if that queue is empty, it steals a task from another queue to execute
- The core-oblivious nature and implicit load balancing through task stealing aids in dynamically changing the number of active threads

Truncated Sprints: Resolving Over-subscription Penalties

- To reduce the number of threads, we just put that thread to sleep once it completes its current task
- The remaining tasks in this worker, if any, are eventually stolen by the other active threads
- Before dequeuing the next task, a worker will check the value of a shared variable; this variable stores the desired number of threads that should be active
- If the worker’s thread id > desired number, the thread sleeps and all of its tasks are stolen by the one remaining thread running on the single core
- This main thread also edits the value of the shared variable

Truncated Sprints: Resolving Over-subscription Penalties

- The new system does not completely replace the older thread migration and core disabling mechanism
- The older mechanism is required in the case when a worker thread that must be put to sleep is running a long task and the task must be suspended and migrated to avoid overheating
- Workload texture is modified to test the new mechanism



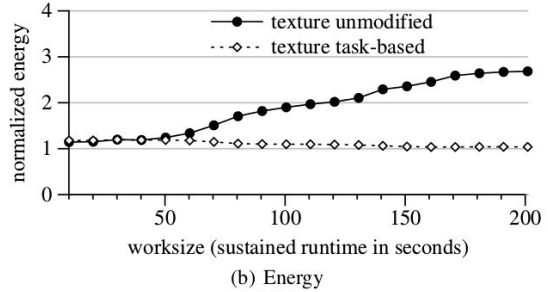
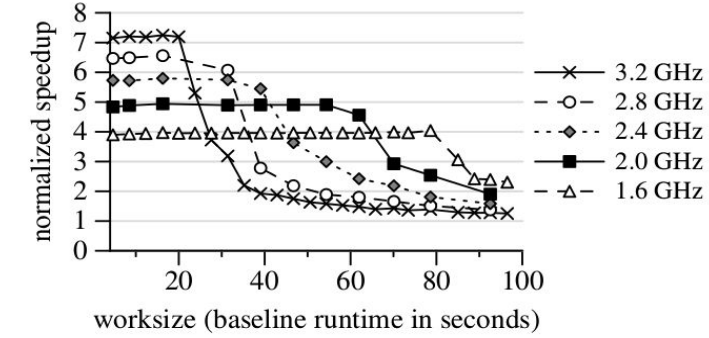


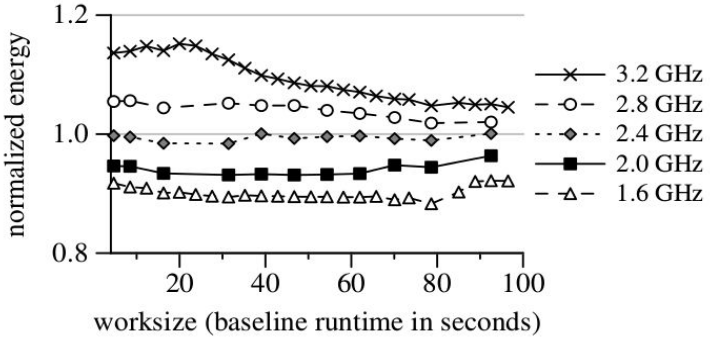
Figure 8. Speedup and energy comparison of the unmodified threaded and task-based implementations of `texture`.

Truncated Sprints: Pacing

- Sprint pacing is a technique that determines what is the optimum frequency for a particular workload so that we get the best responsiveness
- For short durations, maximum intensity sprint achieves best responsiveness
- For large computations, responsiveness is not better than sustainable mode
- For intermediate computation lengths, the optimal mode is not always maximum sprint intensity (just as for humans)



(a) Speedup



(b) Energy

Figure 9. Speedup and energy versus size of computation for sprinting with four cores at different differences.

Truncated Sprints: Pacing

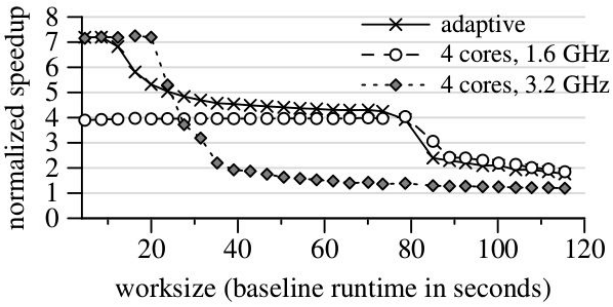
- The responsiveness gain by doubling frequency is at most 2x
- But max. Sprint duration @1.6GHz is 6.3x that @3.2GHz
- A 1.6GHz sprint can complete over 3x more work
- Reasons:
  - Low frequency and voltage use less thermal capacitance per unit work
  - Longer duration of the sprint allows heat to be dissipated to ambient during the sprint
  - Max. intensity sprints do not fully exploit thermal capacitance due to lateral heat conduction delay; low-intensity sprints utilise more thermal capacitance by allowing time for heat to spread

Truncated Sprints: Pacing policy

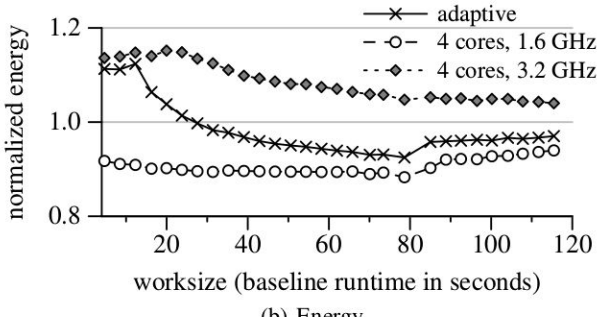
- Most critical factor in sprint pacing policy is length of computation
- Two approaches considered;
  - Predictive sprinting
  - Adaptive sprinting
- Predictive Sprinting (Static)
  - Predict the length of computation using support from h/w, OS or application
  - Determine the appropriate sprint frequency
- Adaptive Sprinting (Dynamic)
  - Sprint pacing policy dynamically adapts to requirement to get the best-case benefit for short computations
  - Moves to a less intense sprint mode to extend the duration of sprint for longer computations where it improves responsiveness

Truncated Sprints: Pacing policy

- Experiment to test adaptive pacing policy
- Sprint policy: Sprint at full intensity until half the thermal capacitance is used, then switch to lowest frequency (no. of cores remains constant)



(a) Speedup

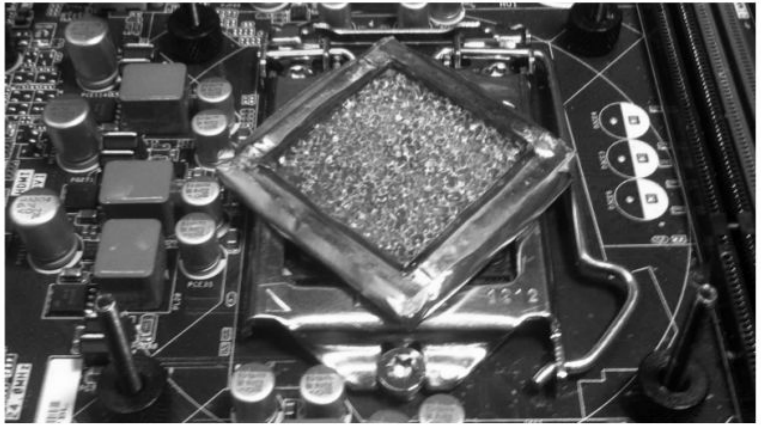


(b) Energy

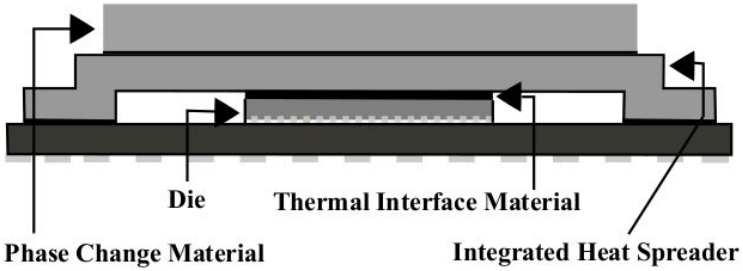
Figure 11. Speedup and energy for sprinting based on an adaptive allocation of sprint budget between the most responsive and most energy efficient schemes.

Extending Sprints: Latent Heat

- Specific heat vs latent heat
- Heat absorbed during phase change does not contribute to increase in temperature
- Thus more time can be gained before temperature reaches T(max)
- One gram of copper heat spreader can absorb 11.5J of heat, one gram of a phase change material can absorb 200J or more
- Experiment:
  - PCM-Paraffin wax infused in Doucel aluminium and enclosed in copper
  - Sprints with 4 cores @ 1.6GHz with three types of thermal capacitance arrangements: heat spreader alone, empty foam (only aluminium and copper) and PCM (paraffin wax) addition



(a) Phase change material on top of the package



(b) Cross-section of phase change material on the package

Figure 12. Testbed augmented with phase-change material.

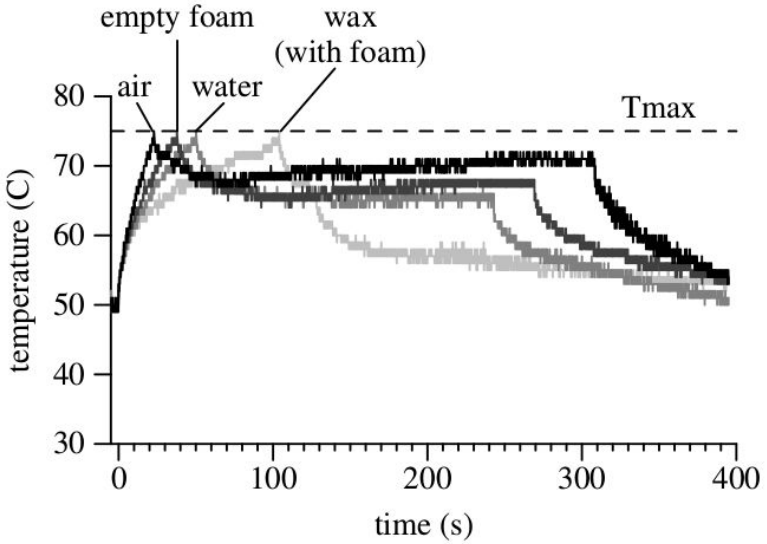


Figure 13. Comparison of sprinting thermal response with and without PCM.

Extending Sprints: Latent Heat

- Adding copper container and aluminium foam alone (empty foam) increases thermal capacitance due to additional specific heat and increases the baseline sprint duration (37s vs 20s)
- Addition of 4g paraffin wax enables the testbed to sprint for 120s - 6x over the baseline
- Reasons:
    - Additional 40s of sprint time due to the melting of paraffin wax (can absorb about 800J of heat)
    - Rest of the time is achieved as heat accumulated is dissipated to the ambient during the phase change
    - Specific heat of paraffin wax also contributes to a lesser extent