Using STT-RAM to Enable Energy-Efficient Near-Threshold Chip Multiprocessors

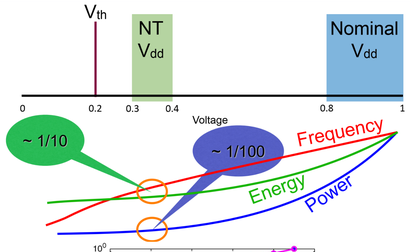
Xiang Pan and Radu Teodorescu
Computer Architecture Research Lab, The Ohio State University

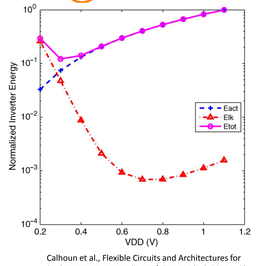
Motivation

Problem: Increasing core counts in CMPs → "Power Wall"

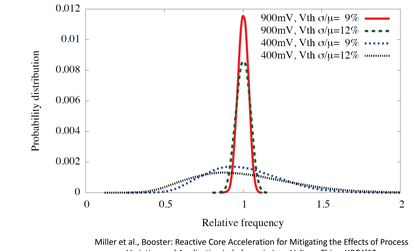
Solution: Near-Threshold Computing

Challenges:
- performance degradation
- amplified process variation
- leakage power dominates

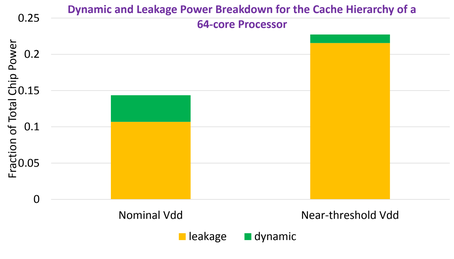




Calhoun et al., Flexible Circuits and Architectures for Ultralow Power, Proceedings of the IEEE (vol. 98, no. 2)



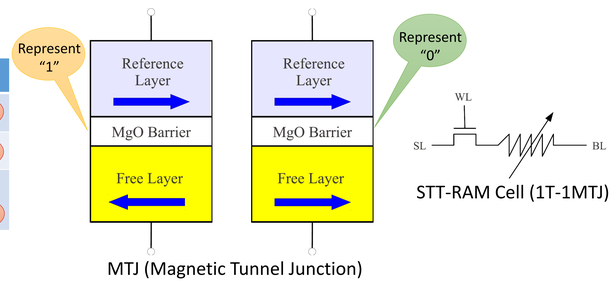
Miller et al., Booster: Reactive Core Acceleration for Mitigating the Effects of Process Variation and Application Imbalance in Low-Voltage Chips, HPCA'12



Approach

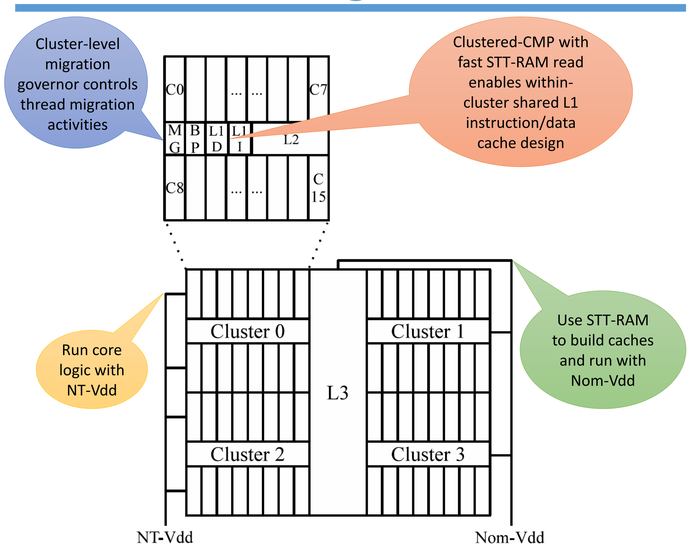
- STT-RAM (Spin Transfer Torque Random Access Memory) has many promising characteristics.
- Great fit for near-threshold designs.

NT-CMP		STT-RAM
High leakage power	⟺	Near-zero leakage power
Low operating frequency	⟺	Long latency writes
Large numbers of cores requiring high cache capacity	⟺	High density (~4x denser than SRAM)



Basic Idea: Use STT-RAM to build NT-CMP caches to save energy and mitigate process variation effects.

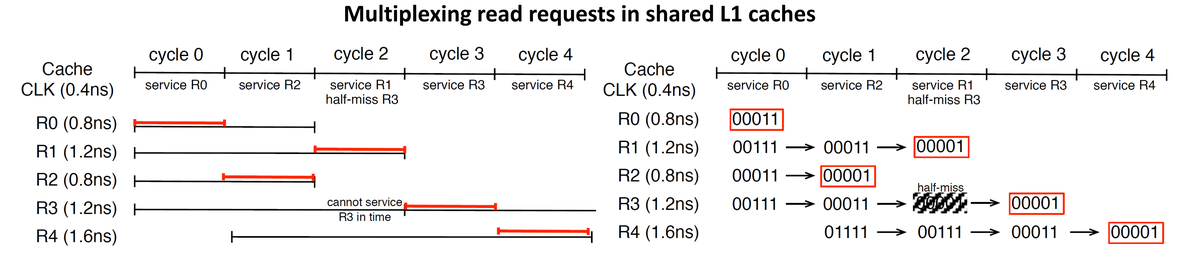
Architecture Design



Methodology

Architecture Configuration	Description
PR-SRAM-NT	NT chip with SRAM private L1(I/D) cache and shared L2/L3 cache
PR-SRAM-Nom	PR-SRAM-NT with nominal Vdd caches
SH-SRAM-Nom	PR-SRAM-Nom with nominal Vdd SRAM shared L1(I/D) cache
SH-STT-NoMig	NT core with nominal Vdd STT-RAM shared L1(I/D) cache and shared L2/L3 cache
SH-STT-Mig(-nK)	SH-STT-NoMig with a nominal Vdd shared branch predictor and performs thread migration every nK cycles
SH-STT-Mig-PRBP	SH-STT-Mig with NT private branch predictors
SH-STT-Mig-Ideal	SH-STT-Mig without any migration cost

Shared Cache Hierarchy

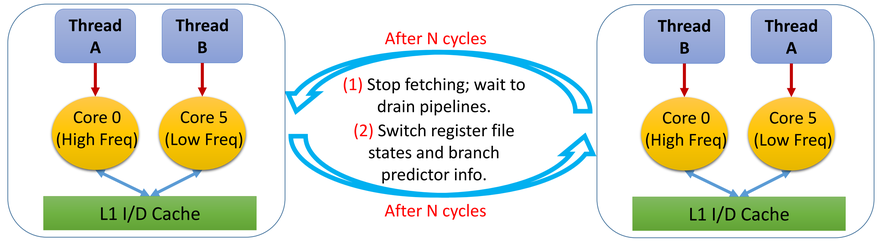


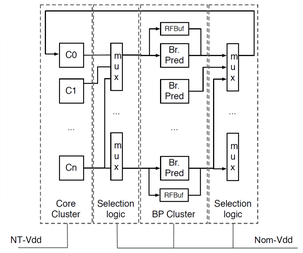
L1 cache much faster than cores → Share L1 within a cluster → Reduce coherence traffic, improving performance

Reduce Core-to-Core Variation Effects

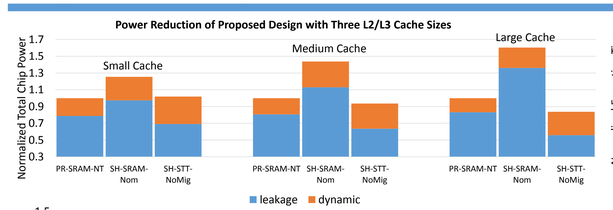
-- Goal: Achieve equal progress on heterogeneous frequency cores.

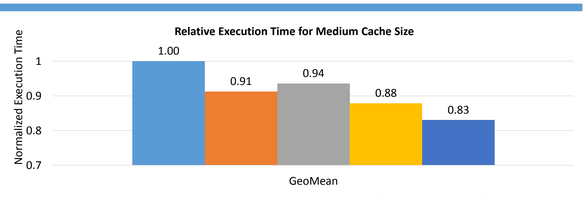
- Based on different frequencies of cores in the same cluster, form "fast-slow" core pairs and migrate threads periodically within each pair.
- Cache data already shared, reducing migration overhead.
- Add shared branch predictor and register file buffer components to the design to further reduce overhead.

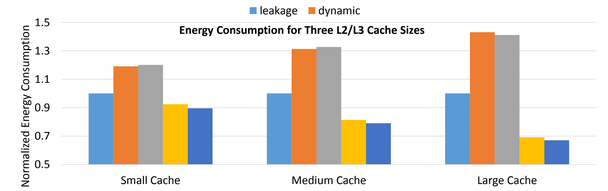




Key Results (Configurations: Small (8MB L2; 24MB L3); Medium (16MB L2; 48MB L3); Large (32MB L2; 96MB L3).)







Cluster Size (# cores)	Shared L1 (I/D) Size	Performance Gain (%)
4	64	4.15
8	128	8.82
16	256	12.15
32	512	-19.08