A Design Methodology for Domain-Optimized Power-Efficient Supercomputing

Marghoob Mohiyuddin, Mark Murphy, Leonid Oliker, John Shalf, John Wawrzynek, Samuel Williams

marghoob@eecs.berkeley.edu

SC09, Nov 19, 2009

Takeaway

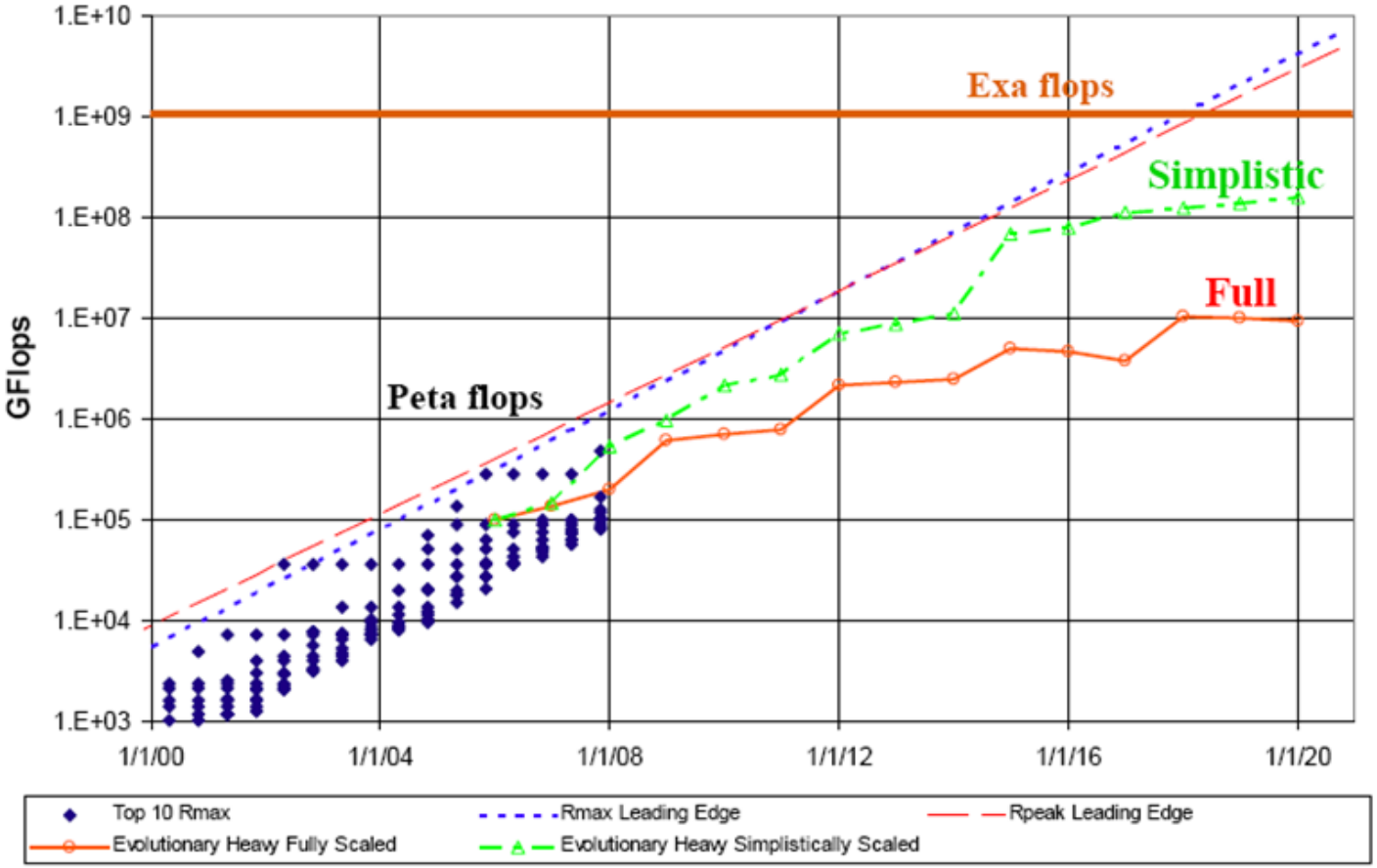
- HW/SW co-tuning as a new approach to HW design
- Applied the new approach to 3 scientific computing kernels and the Stanford Smart Memories multiprocessor
- Results show efficiency improves significantly when HW designed using co-tuning

Outline

Outline

Will exascale happen?

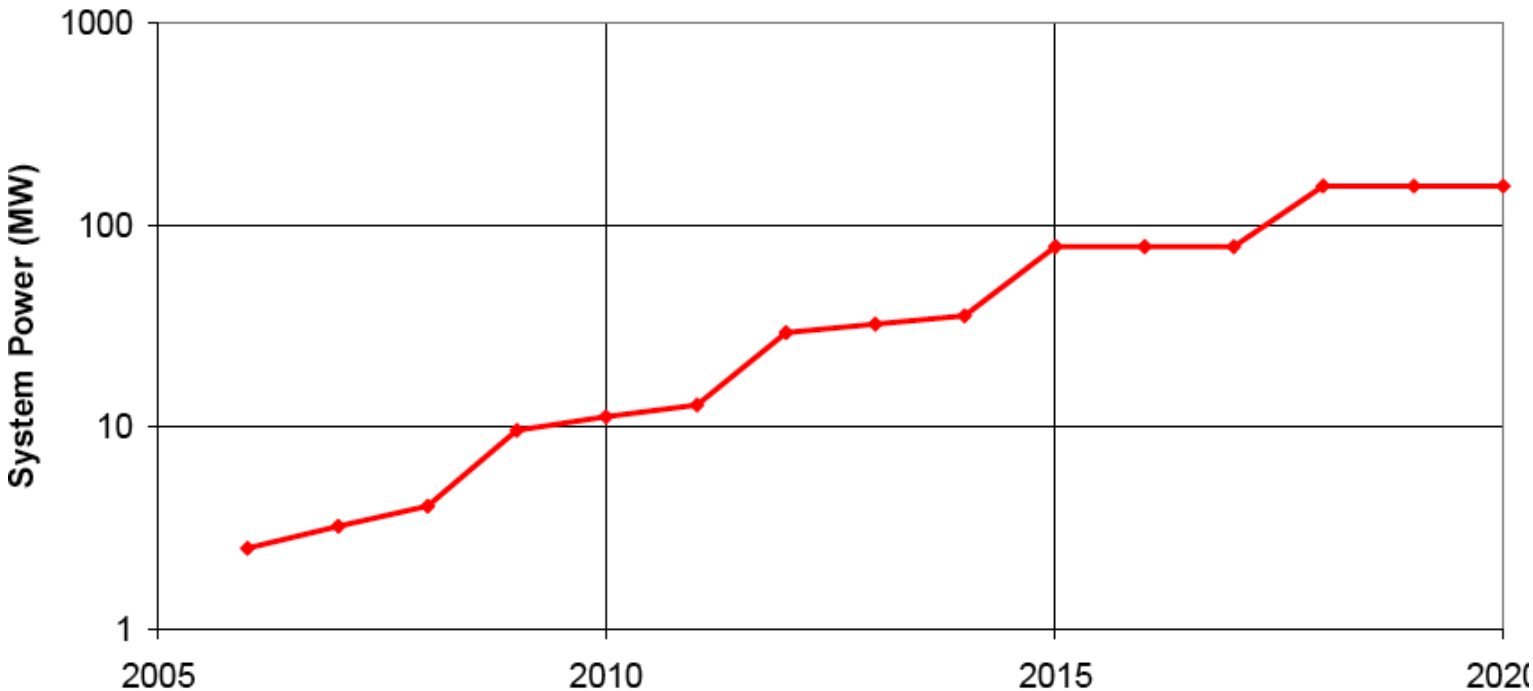
. . . with the current approach



From Peter Kogge, DARPA exascale study

At what cost?

- Power-efficiency not improving at historic rates
- Petaflop systems already draw Megawatts of power
- DARPA exascale study predicts $>$ 100 Megawatts of power for exaflop systems



From Peter Kogge, DARPA exascale study

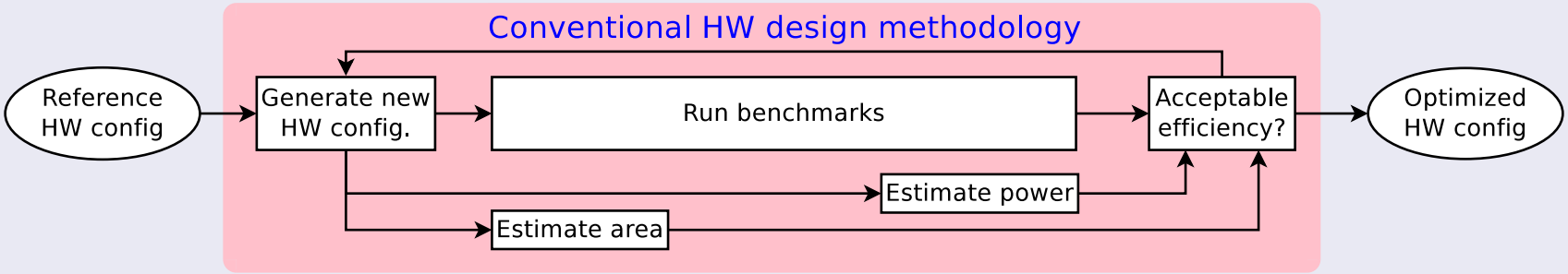
What is wrong with current HW design approaches?

- General-purpose commodity processors in many large machines are power-inefficient
- HW customization improves energy efficiency
  - Simpler cores more power-efficient
  - Intel Core2 sc: 15W@1000 MHz
    Tensilica XTensa DP: .09W@600 MHz

What is wrong with current HW design approaches?

- General-purpose commodity processors in many large machines are power-inefficient
- HW customization improves energy efficiency
  - Simpler cores more power-efficient
  - Intel Core2 sc: 15W@1000 MHz
    Tensilica XTensa DP: .09W@600 MHz

Typical HW design space exploration



- HW config parameters: # cores, cache/local store organization, interconnect, DRAM latency/bandwidth, etc
- Find the right balance of parameters: cores vs. cache, bandwidth vs. peak flop rate
- Benchmarks not optimized for each HW config considered

What is right with current SW tuning?

What is right with current SW tuning?

Answer: auto-tuning

What is right with current SW tuning?

Answer: auto-tuning

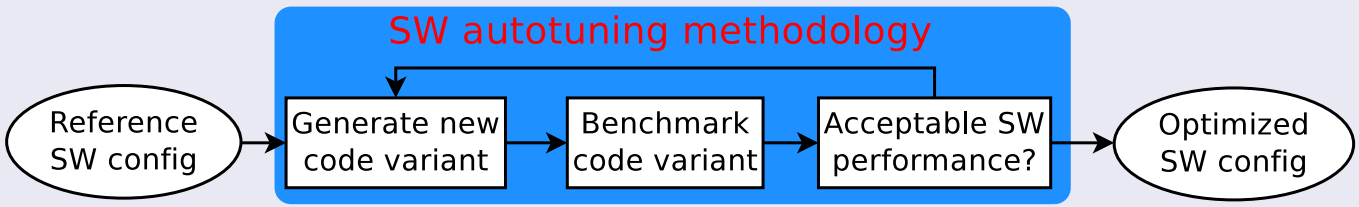
- Automate the process of optimizing SW for a variety of architectures
  - Assumption: architectures evolve $\Rightarrow$ optimizations still valid
- Key to portable high-performance libraries: ATLAS, OSKI, FFTW, SPIRAL

What is right with current SW tuning?

Answer: auto-tuning

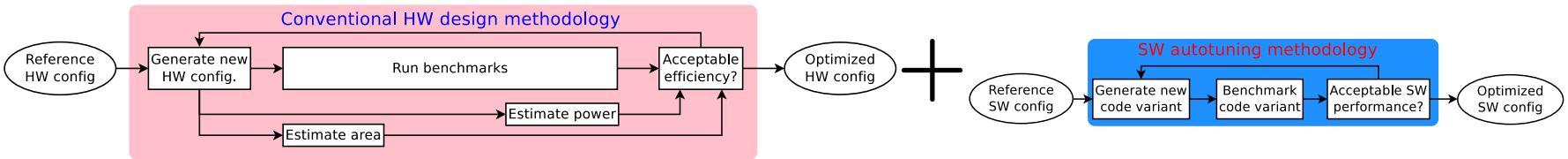
- Automate the process of optimizing SW for a variety of architectures
  - Assumption: architectures evolve ⇒ optimizations still valid
- Key to portable high-performance libraries: ATLAS, OSKI, FFTW, SPIRAL

Conventional SW autotuning

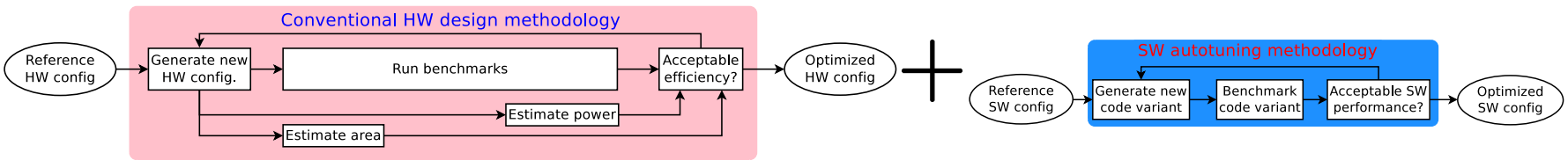


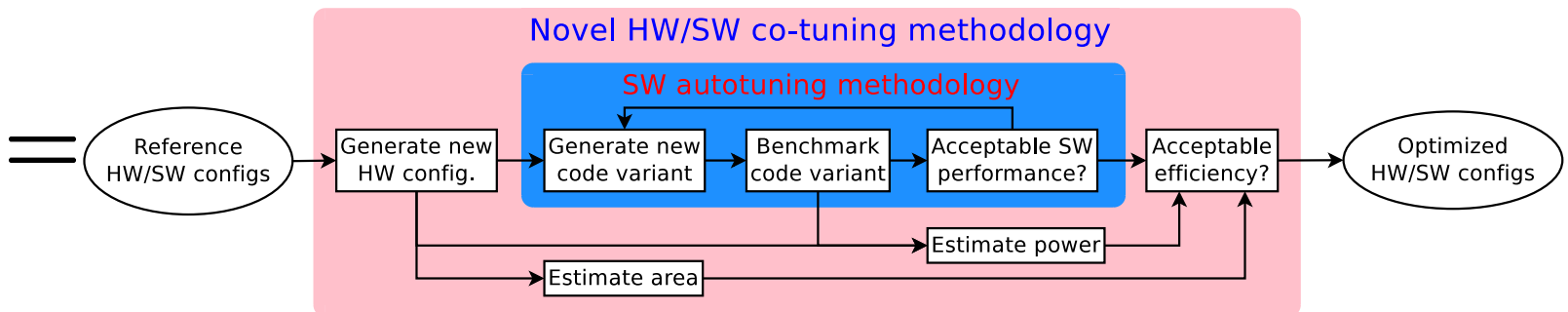
- SW config parameters: register/cache block sizes, loop unroll factor, data structures, algorithms, etc
- Source code generators + parameterized routines + search heuristic
- Offline (install time), runtime tuning

HW/SW Co-tuning: The solution

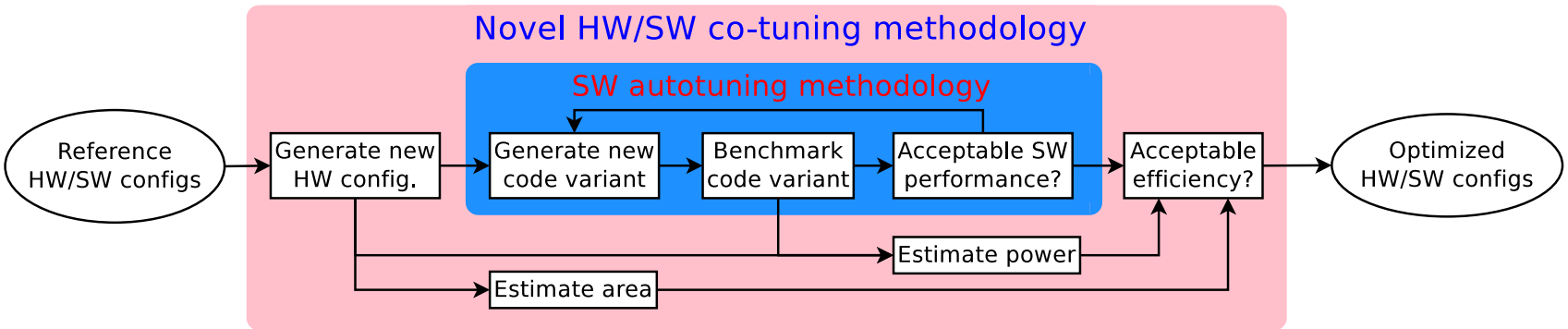


HW/SW Co-tuning: The solution



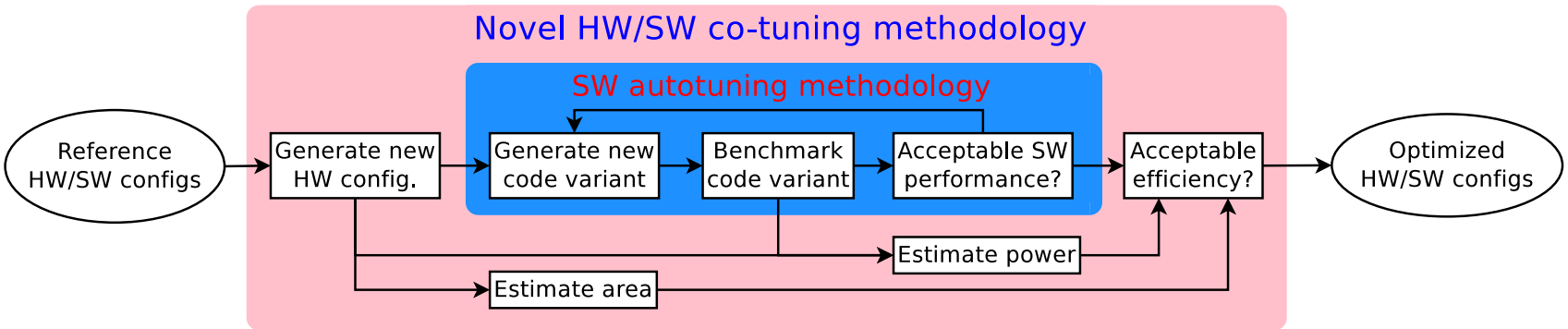


HW/SW Co-tuning



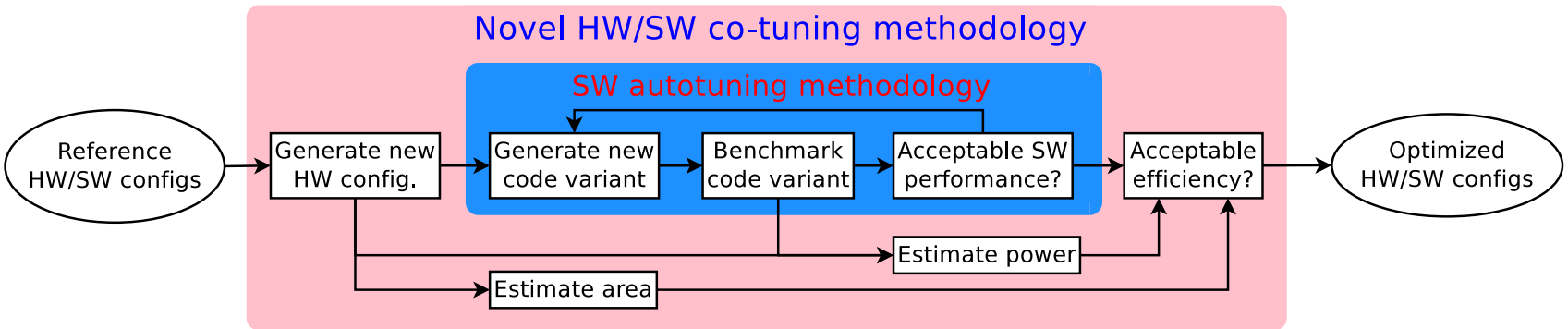
- Key idea: include SW autotuning in the loop for HW design
- A rigorous systematic approach to HW design
- This is also the approach taken in Green Flash where the target application is climate modeling
  - Our results affirm the effectiveness of co-tuning

HW/SW Co-tuning



- Key idea: include SW autotuning in the loop for HW design
- A rigorous systematic approach to HW design
- This is also the approach taken in Green Flash where the target application is climate modeling
  - Our results affirm the effectiveness of co-tuning

HW/SW Co-tuning

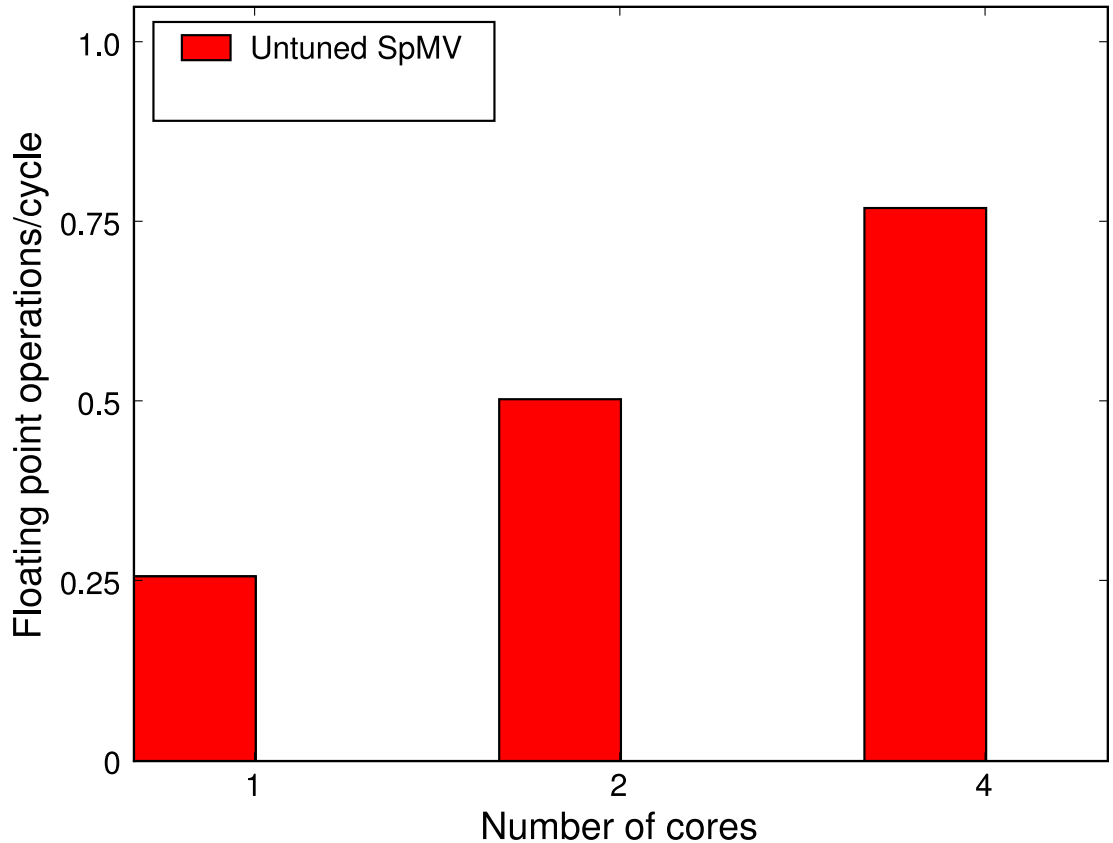


- Key idea: include SW autotuning in the loop for HW design
- A rigorous systematic approach to HW design
- This is also the approach taken in Green Flash where the target application is climate modeling
  - Our results affirm the effectiveness of co-tuning

A simple example

Sparse matrix vector multiply performance (121K rows, nnz/row=27.3) on Stanford Smart Memories multiprocessor

(DRAM bandwidth = 1.6 GB/s, cache/core = 64 KB)

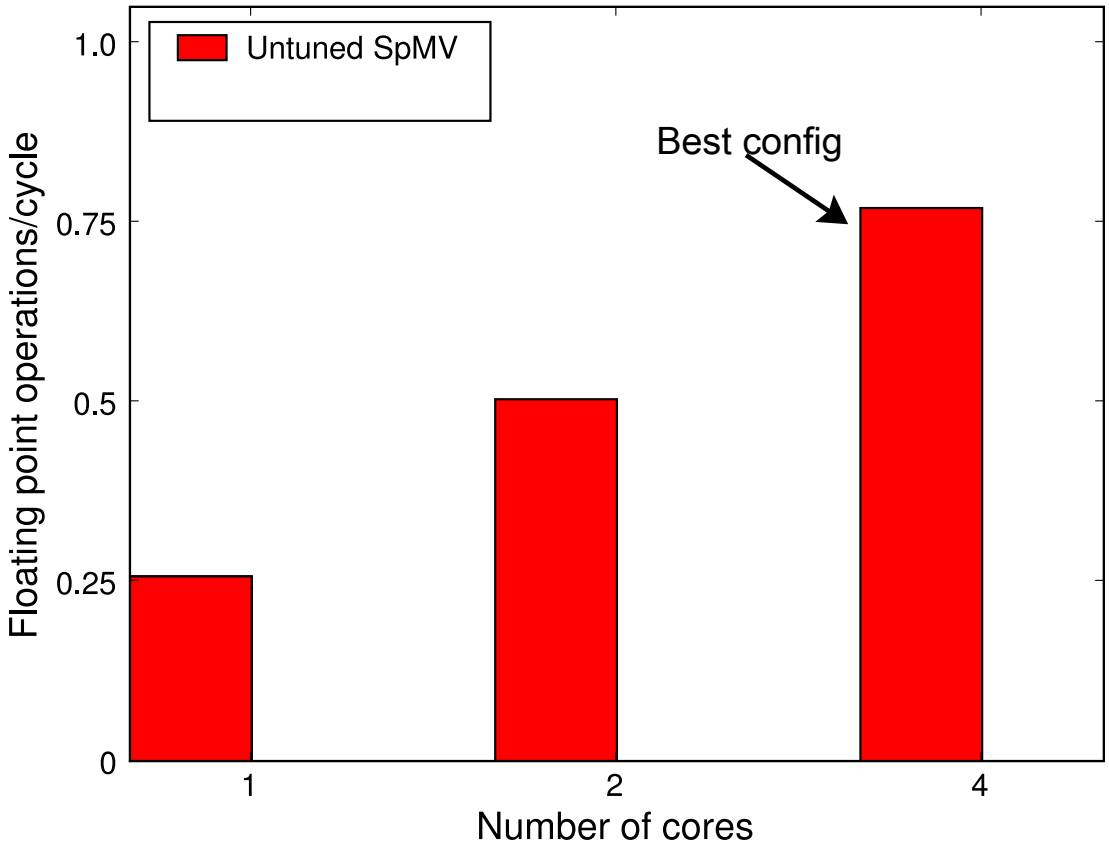


- For tuned SpMV, best #cores = 2 (same performance as 4 cores with half the area)
- For untuned SpMV, best #cores = 4 ⇒ overdesign

A simple example

Sparse matrix vector multiply performance (121K rows, nnz/row=27.3) on Stanford Smart Memories multiprocessor

(DRAM bandwidth = 1.6 GB/s, cache/core = 64 KB)

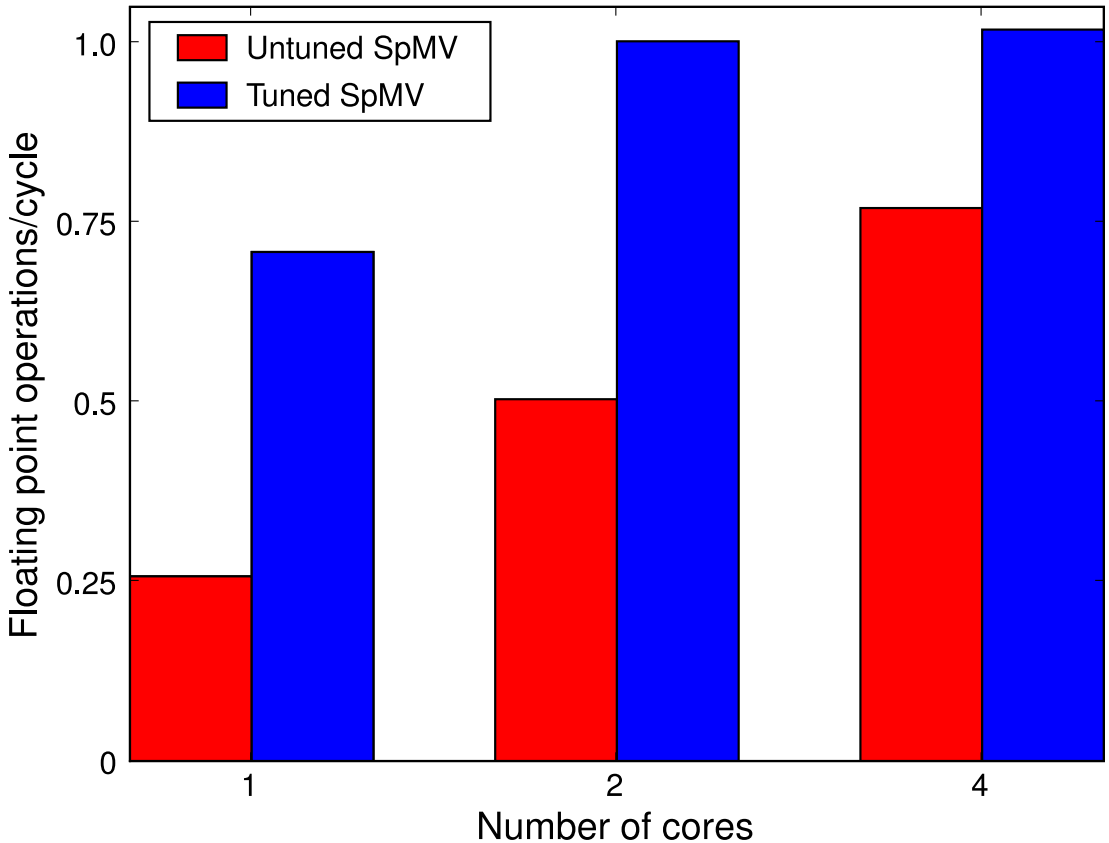


- For tuned SpMV, best #cores = 2 (same performance as 4 cores with half the area)
- For untuned SpMV, best #cores = 4 ⇒ overdesign

A simple example

Sparse matrix vector multiply performance (121K rows, nnz/row=27.3) on Stanford Smart Memories multiprocessor

(DRAM bandwidth = 1.6 GB/s, cache/core = 64 KB)

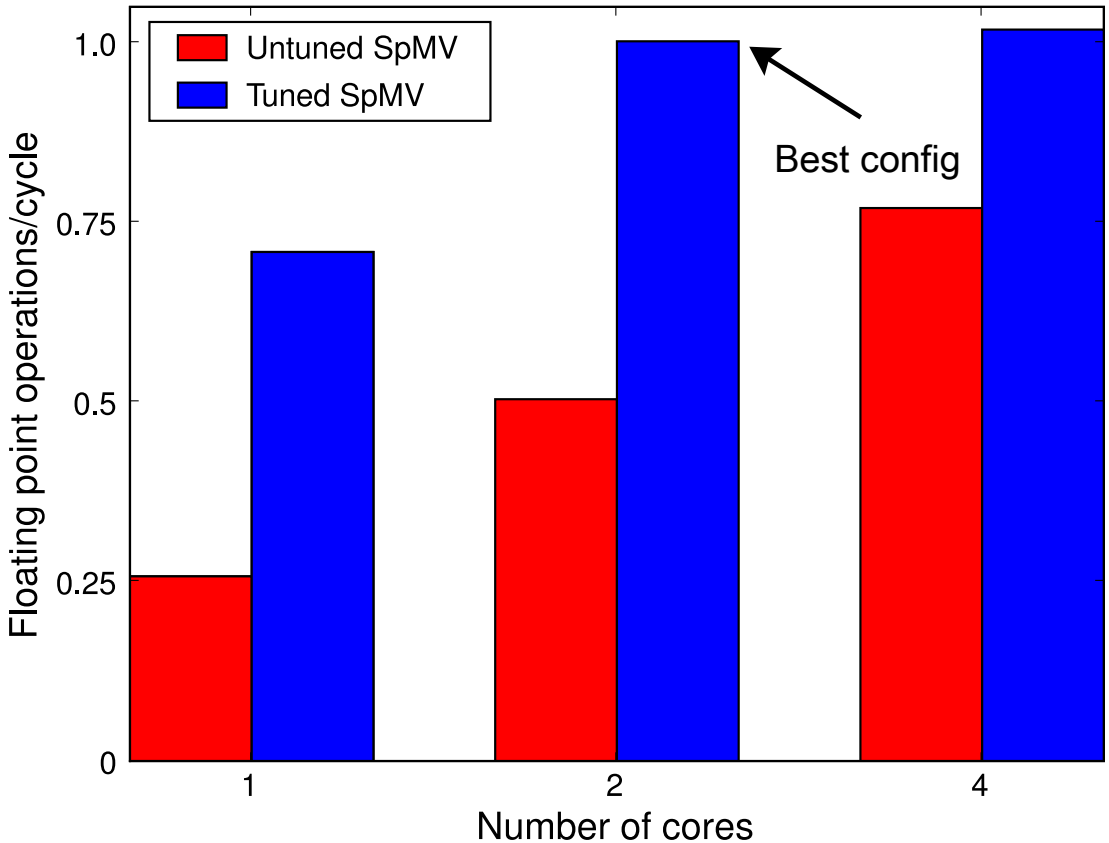


- For tuned SpMV, best #cores = 2 (same performance as 4 cores with half the area)
- For untuned SpMV, best #cores = 4 ⇒ overdesign

A simple example

Sparse matrix vector multiply performance (121K rows, nnz/row=27.3) on Stanford Smart Memories multiprocessor

(DRAM bandwidth = 1.6 GB/s, cache/core = 64 KB)



- For tuned SpMV, best #cores = 2 (same performance as 4 cores with half the area)
- For untuned SpMV, best #cores = 4 ⇒ overdesign

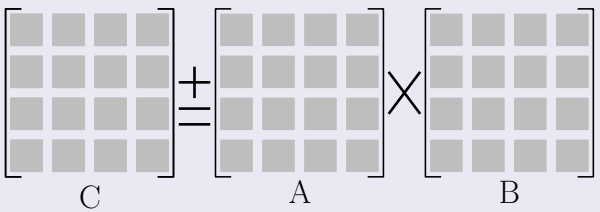
Outline

Application of HW/SW co-tuning

- Software: 3 kernels from scientific computing:
  - Dense matrix matrix multiplication (dense linear algebra)
  - 7pt stencil operator (heat equation PDE)
  - Sparse matrix vector multiplication (sparse linear algebra)
  - Varying computational characteristics
    $\Rightarrow$ pull HW parameters in diff. directions
  - Success of co-tuning demonstrated by application on multiple kernels
- Hardware: Stanford Smart Memories multiprocessor
  - Multiprocessor using Tensilica cores
  - Analogous to the Green Flash design which uses the same cores

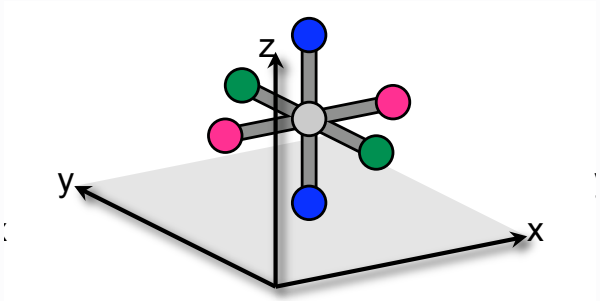
The Kernels

Dense matrix matrix multiplication (GEMM)



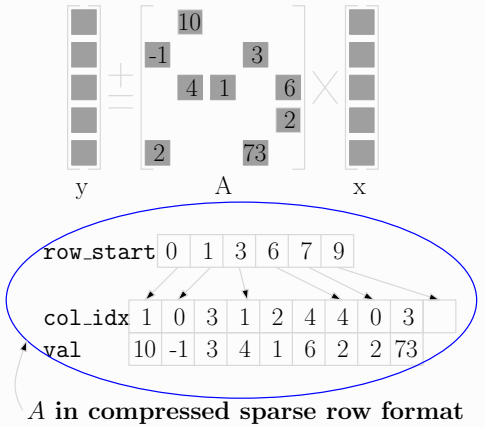
- Dense linear algebra
- High computational intensity
- Tuned code gets close to machine peak
- More cores ⇒ better performance
- $2N^3$ flops for multiplying 2 $N \times N$ matrices
- $12N^2$ bytes compulsory memory traffic

7-pt stencil operator on 3D grid



- Explicit finite-difference method for the heat equation
- Low computational intensity, regular memory accesses
- More bandwidth ⇒ better performance
- $8N^3$ flops on an $N \times N \times N$ grid
- $8N^3$ bytes compulsory memory traffic

Sparse matrix vector multiplication (SpMV)

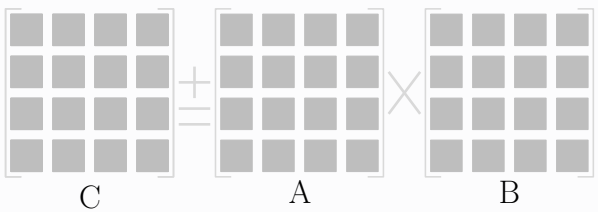


A in compressed sparse row format

- Used in PDEs, sparse solvers
- Low computational intensity, irregular memory accesses
- More bandwidth ⇒ better performance
- $2 \cdot nnz$ flops (nnz = # nonzeros)
- $4 \cdot nnz$ bytes compulsory memory traffic

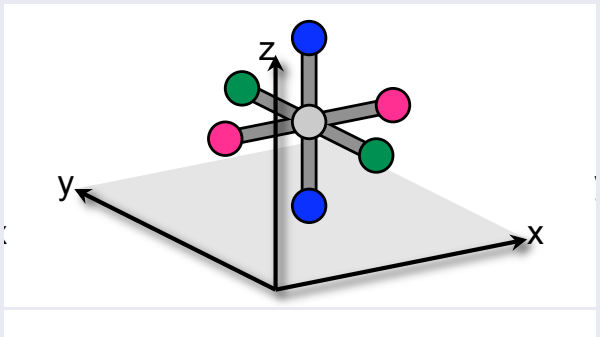
The Kernels

Dense matrix matrix multiplication (GEMM)



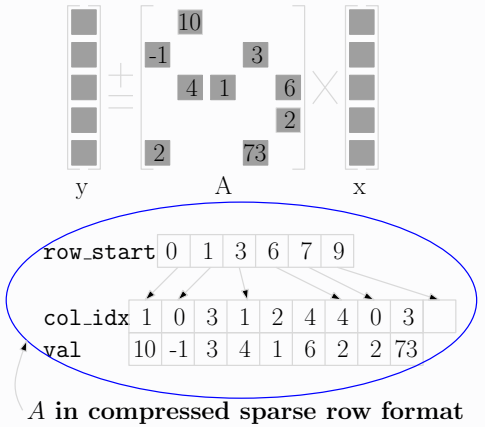
- Dense linear algebra
- High computational intensity
- Tuned code gets close to machine peak
- More cores ⇒ better performance
- $2N^3$ flops for multiplying 2 $N \times N$ matrices
- $12N^2$ bytes compulsory memory traffic

7-pt stencil operator on 3D grid



- Explicit finite-difference method for the heat equation
- Low computational intensity, regular memory accesses
- More bandwidth ⇒ better performance
- $8N^3$ flops on an $N \times N \times N$ grid
- $8N^3$ bytes compulsory memory traffic

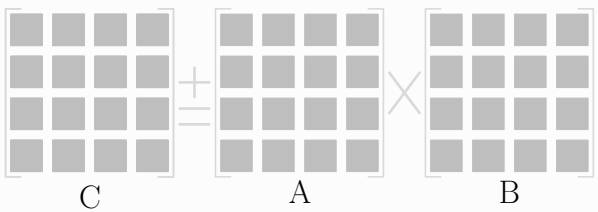
Sparse matrix vector multiplication (SpMV)



- Used in PDEs, sparse solvers
- Low computational intensity, irregular memory accesses
- More bandwidth ⇒ better performance
- $2 \cdot nnz$ flops (nnz = # nonzeros)
- $4 \cdot nnz$ bytes compulsory memory traffic

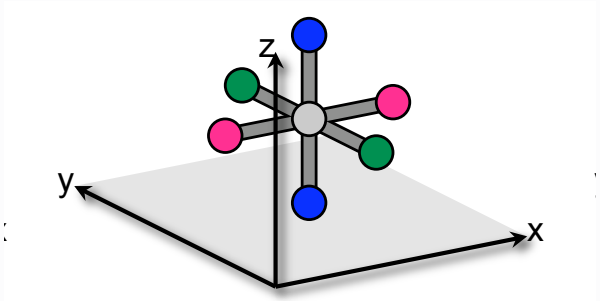
The Kernels

Dense matrix matrix multiplication (GEMM)



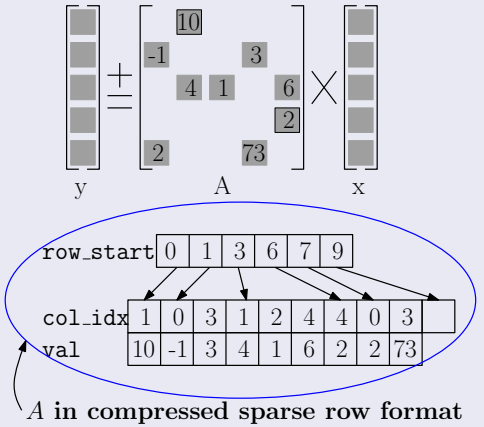
- Dense linear algebra
- High computational intensity
- Tuned code gets close to machine peak
- More cores ⇒ better performance
- $2N^3$ flops for multiplying 2 $N \times N$ matrices
- $12N^2$ bytes compulsory memory traffic

7-pt stencil operator on 3D grid



- Explicit finite-difference method for the heat equation
- Low computational intensity, regular memory accesses
- More bandwidth ⇒ better performance
- $8N^3$ flops on an $N \times N \times N$ grid
- $8N^3$ bytes compulsory memory traffic

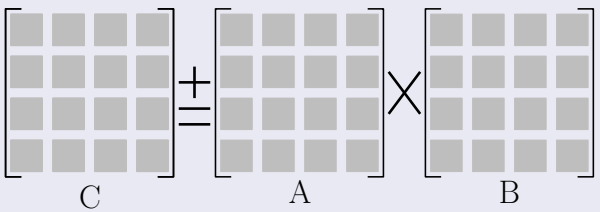
Sparse matrix vector multiplication (SpMV)



- Used in PDEs, sparse solvers
- Low computational intensity, irregular memory accesses
- More bandwidth ⇒ better performance
- $2 \cdot nnz$ flops (nnz = # nonzeros)
- $4 \cdot nnz$ bytes compulsory memory traffic

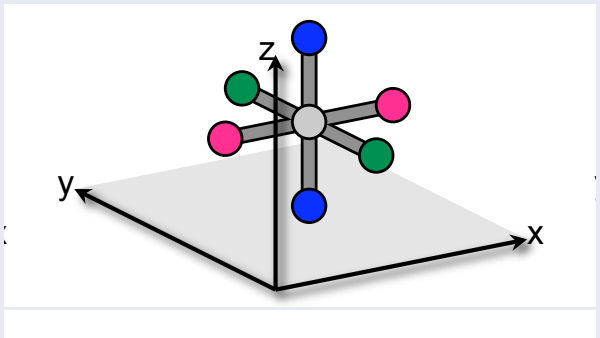
The Kernels

Dense matrix matrix multiplication (GEMM)



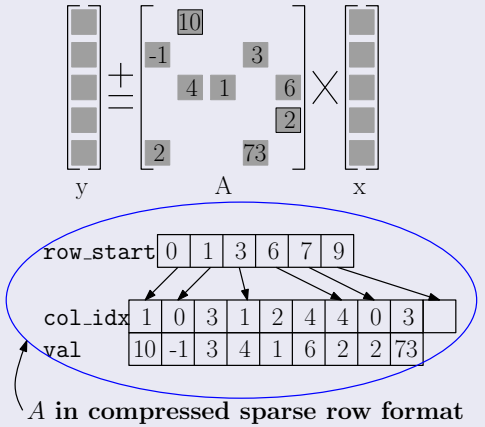
- Dense linear algebra
- High computational intensity
- Tuned code gets close to machine peak
- More cores ⇒ better performance
- $2N^3$ flops for multiplying 2 $N \times N$ matrices
- $12N^2$ bytes compulsory memory traffic

7-pt stencil operator on 3D grid



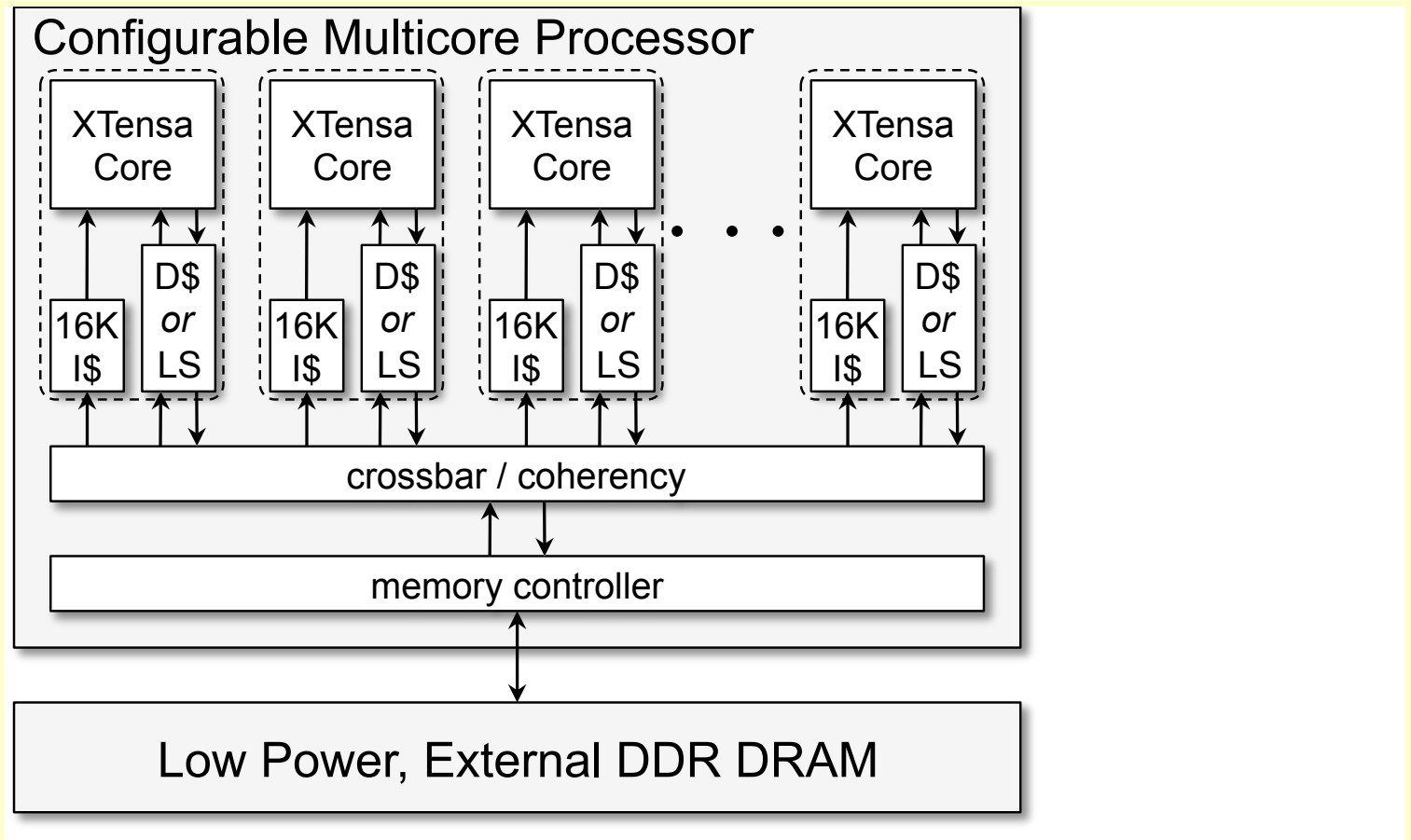
- Explicit finite-difference method for the heat equation
- Low computational intensity, regular memory accesses
- More bandwidth ⇒ better performance
- $8N^3$ flops on an $N \times N \times N$ grid
- $8N^3$ bytes compulsory memory traffic

Sparse matrix vector multiplication (SpMV)



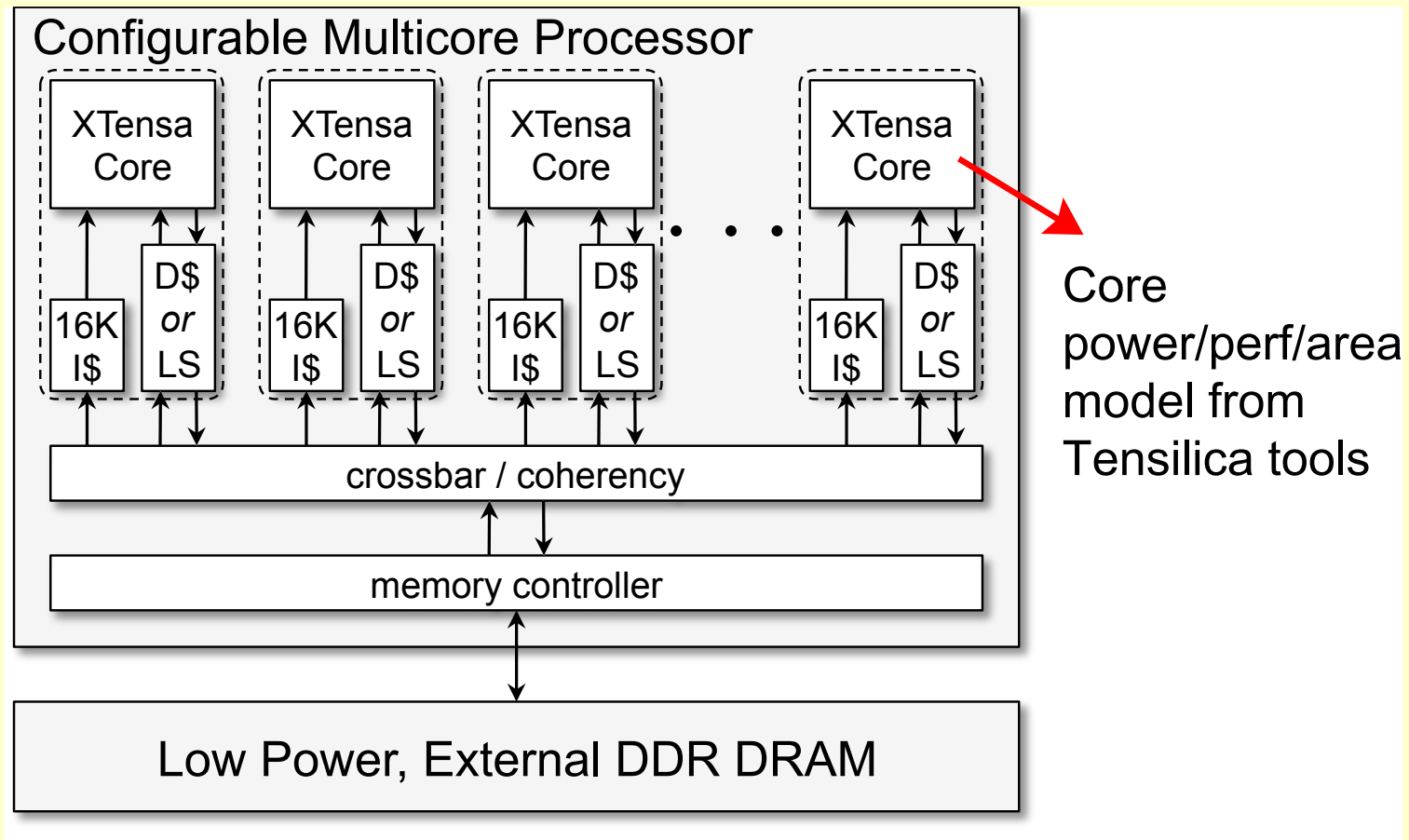
- Used in PDEs, sparse solvers
- Low computational intensity, irregular memory accesses
- More bandwidth ⇒ better performance
- $2 \cdot nnz$ flops (nnz = # nonzeros)
- $4 \cdot nnz$ bytes compulsory memory traffic

The Hardware: Stanford Smart Memories Multiprocessor



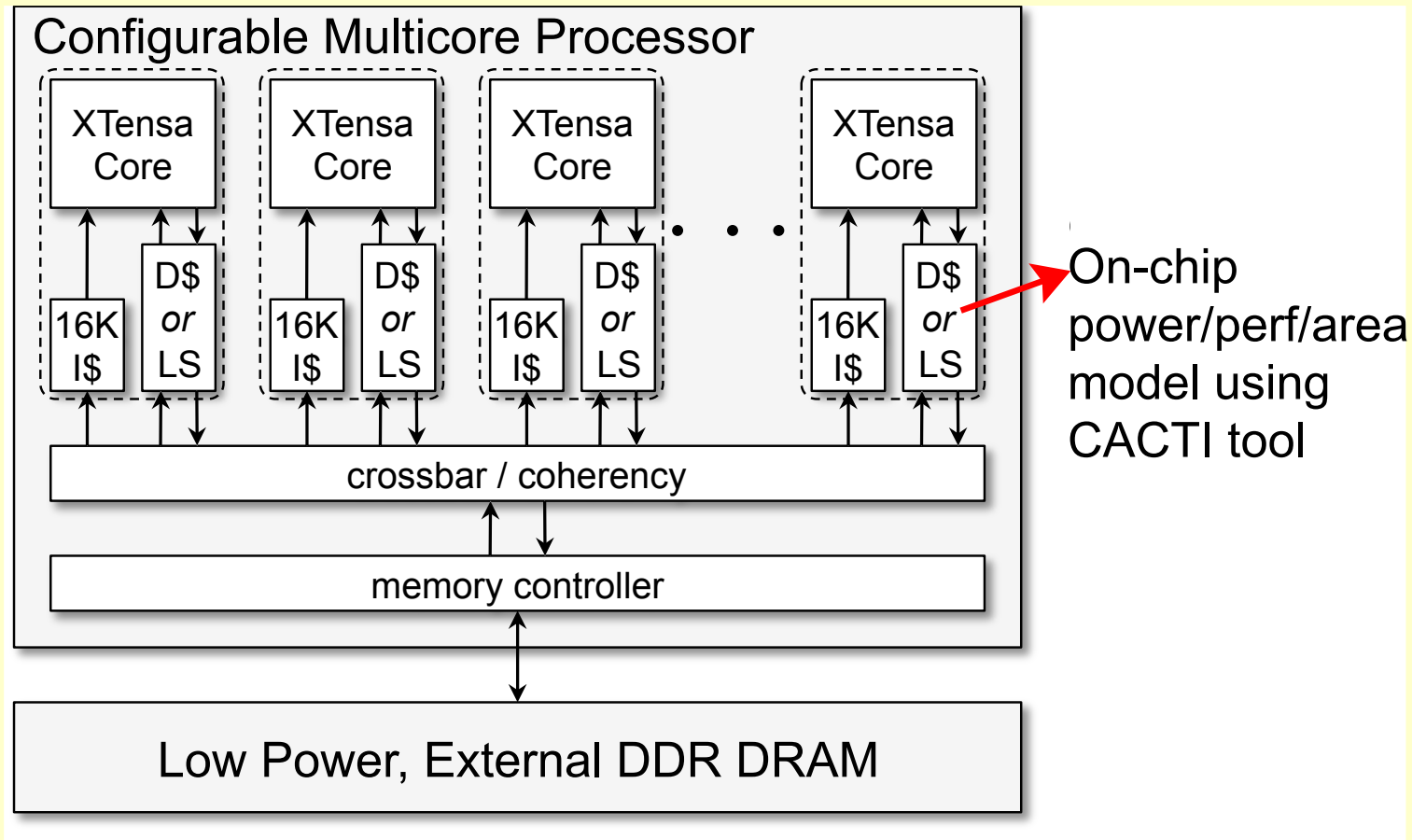
- Each core has a single-precision FPU
- Constant 'area' of 35mm^2 added to include the impact of DRAM cost

The Hardware: Stanford Smart Memories Multiprocessor



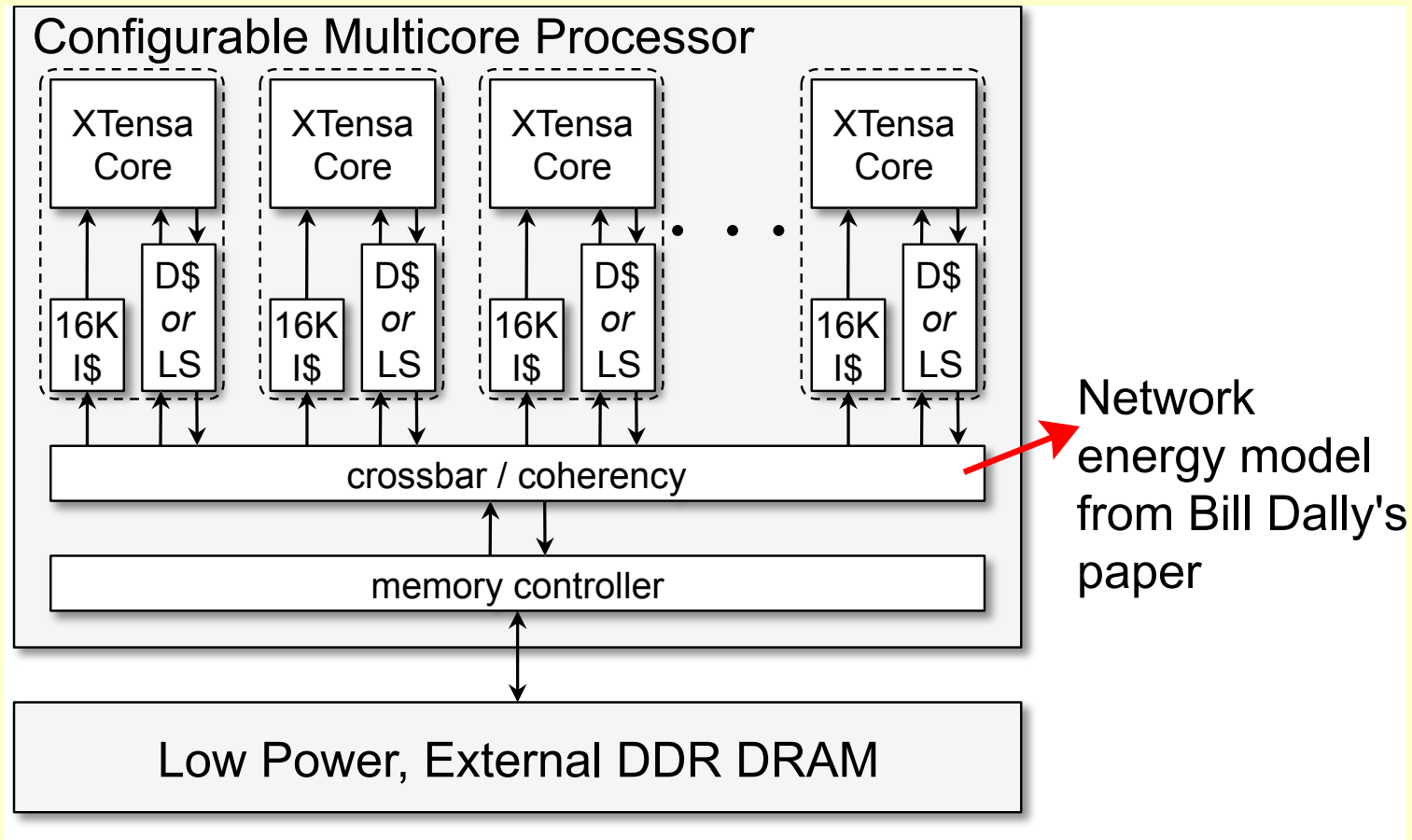
- Each core has a single-precision FPU
- Constant 'area' of 35mm^2 added to include the impact of DRAM cost

The Hardware: Stanford Smart Memories Multiprocessor



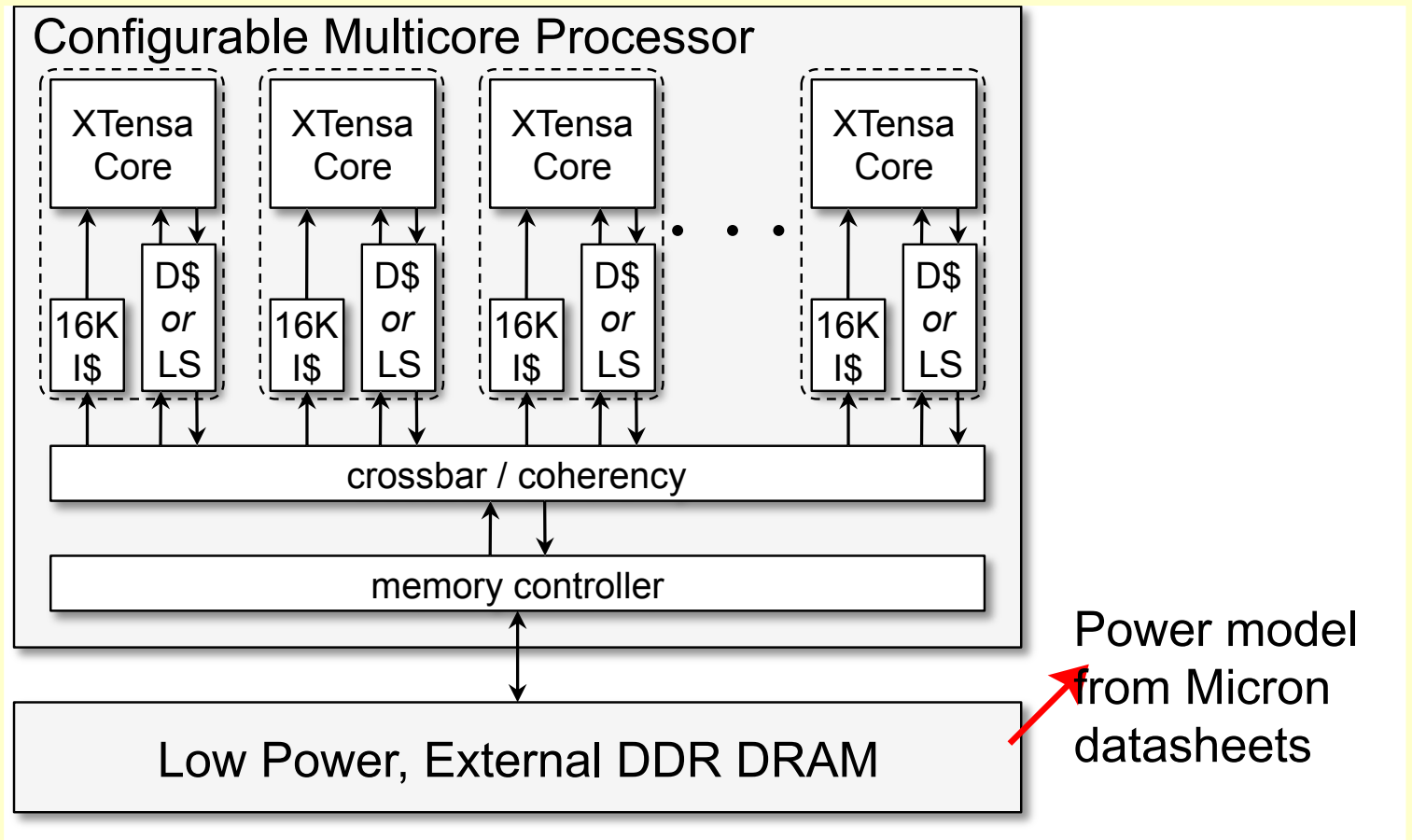
- Each core has a single-precision FPU
- Constant 'area' of 35mm^2 added to include the impact of DRAM cost

The Hardware: Stanford Smart Memories Multiprocessor



- Each core has a single-precision FPU
- Constant 'area' of 35mm^2 added to include the impact of DRAM cost

The Hardware: Stanford Smart Memories Multiprocessor



- Each core has a single-precision FPU
- Constant 'area' of 35mm^2 added to include the impact of DRAM cost

Hardware Parameters

- **Fixed:**
  - Core: single-issue, 500 MHz
  - Cache/local store: 16 KB I-cache, cache associativity = 4, linesize = 64 bytes
  - DRAM: latency = 100 core cycles
- **Variable:**
  - # cores: 1/4/16
  - On-chip data memory type: cache/local store
  - Cache/local store per core: 16, 32, 64, 128 KB
  - DRAM bandwidth: 0.8, 1.6, 3.2 GB/s
  - 72 HW configs
- **Baseline config:** Fastest HW
  - On-chip memory type: cache
  - Cache per core: 128 KB
  - DRAM bandwidth: 3.2 GB/s

Hardware Parameters

- **Fixed:**
  - Core: single-issue, 500 MHz
  - Cache/local store: 16 KB I-cache, cache associativity = 4, linesize = 64 bytes
  - DRAM: latency = 100 core cycles
- **Variable:**
  - # cores: 1/4/16
  - On-chip data memory type: cache/local store
  - Cache/local store per core: 16, 32, 64, 128 KB
  - DRAM bandwidth: 0.8, 1.6, 3.2 GB/s
  - 72 HW configs
- **Baseline config:** Fastest HW
  - On-chip memory type: cache
  - Cache per core: 128 KB
  - DRAM bandwidth: 3.2 GB/s

Outline

Optimized Metrics

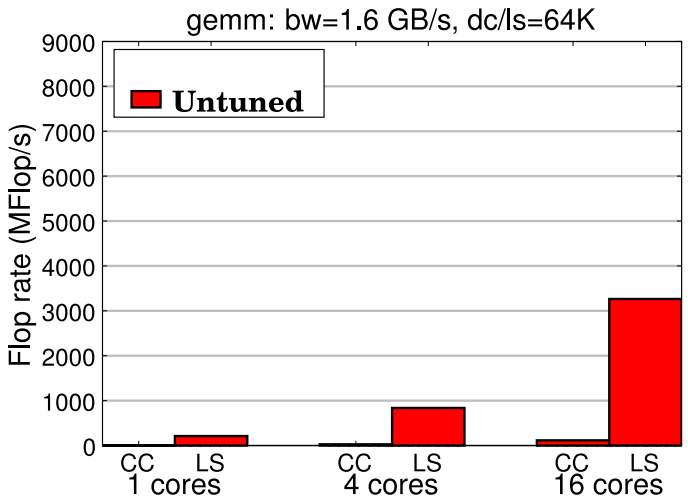
- Focus on scientific computing apps running on large-scale systems
  - Emphasize node efficiency instead of node performance
- Power efficiency (MFlops/Watt)
  - Running costs
  - Maximize performance given a power budget
- Area efficiency (MFlops/mm^2)
  - System cost, reliability dependent on area
  - Maximize performance given an area budget
- Power efficiency, area efficiency can result in different optimal HW config
- In general, would want to optimize a combination of power-, area-efficiencies

Effect of SW tuning on performance

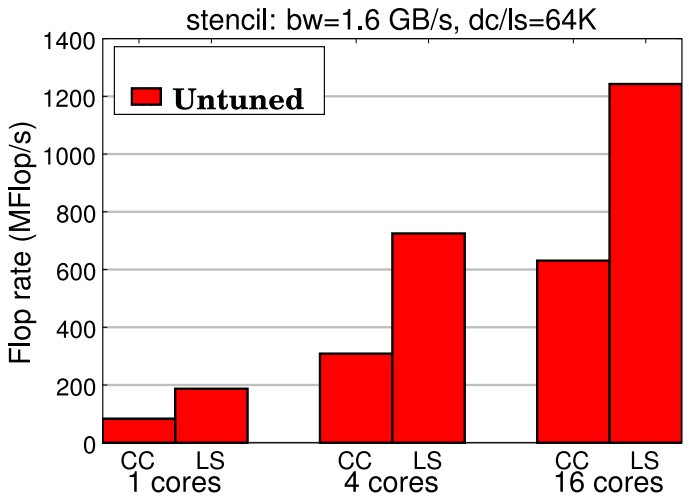
DRAM bw = 1.6 GB/s, D-cache/local store = 64 KB

(CC=cache, LS=local store)

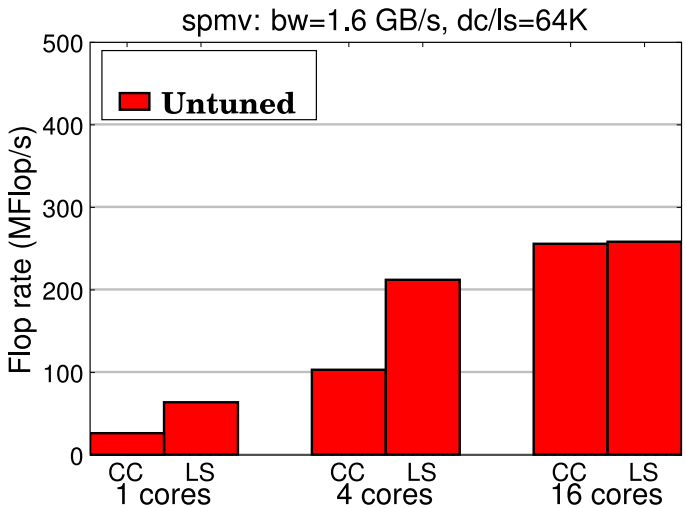
GEMM



Stencil



SpMV



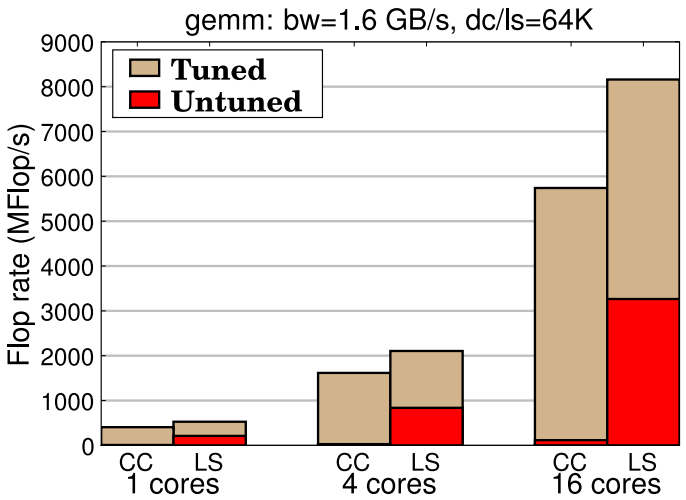
- GEMM gains a lot from tuning
- Software-managed caches get better performance
- Bandwidth-saturation for stencil and SpMV

Effect of SW tuning on performance

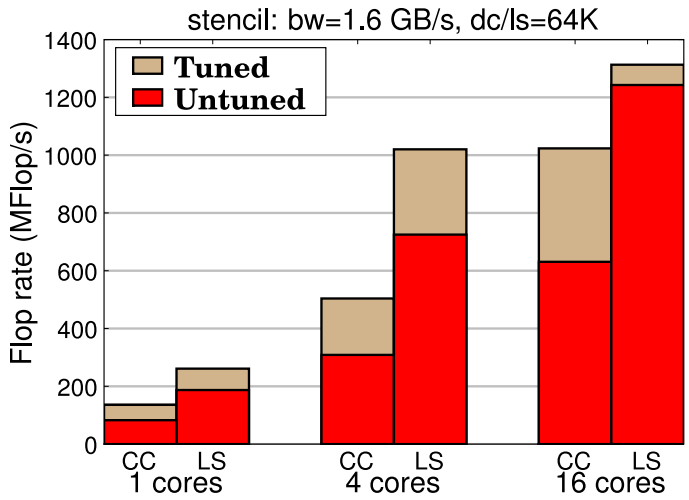
DRAM bw = 1.6 GB/s, D-cache/local store = 64 KB

(CC=cache, LS=local store)

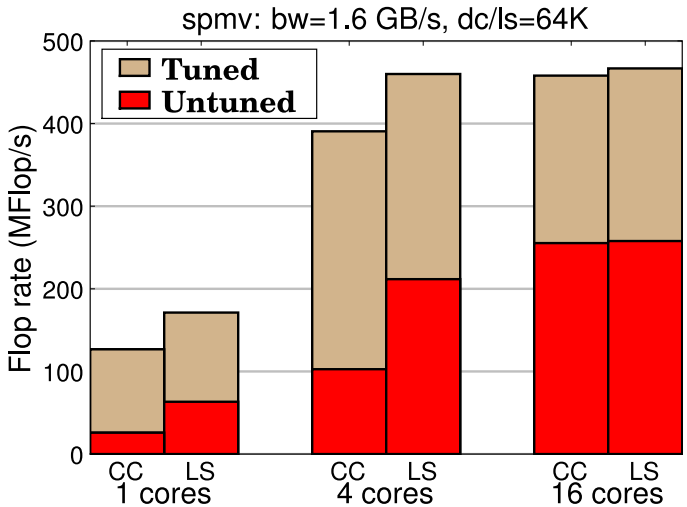
GEMM



Stencil



SpMV



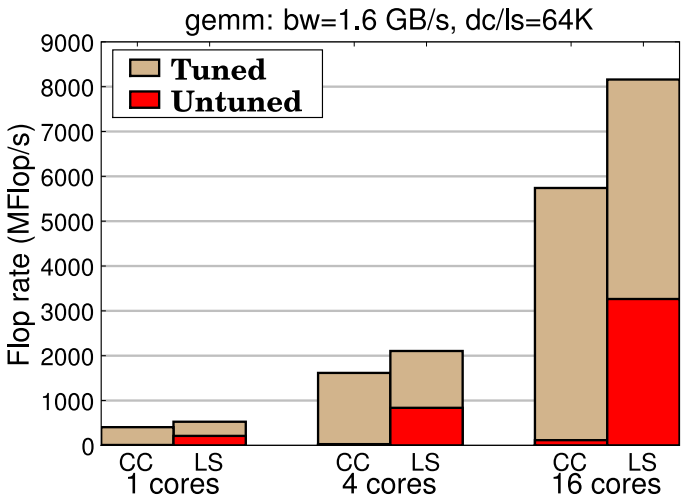
- GEMM gains a lot from tuning
- Software-managed caches get better performance
- Bandwidth-saturation for stencil and SpMV

Effect of SW tuning on performance

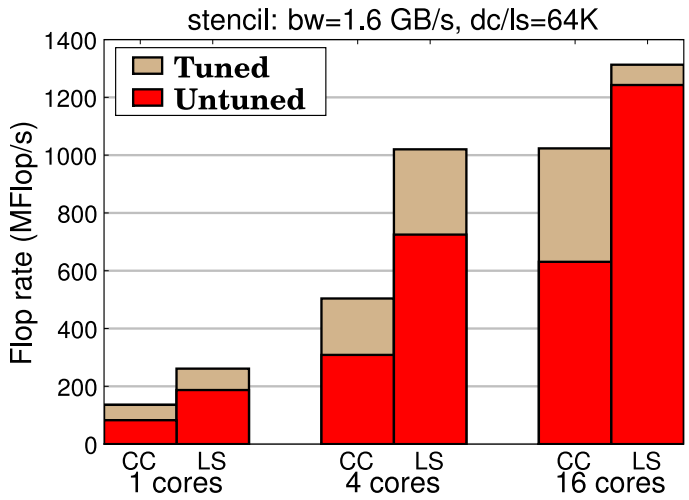
DRAM bw = 1.6 GB/s, D-cache/local store = 64 KB

(CC=cache, LS=local store)

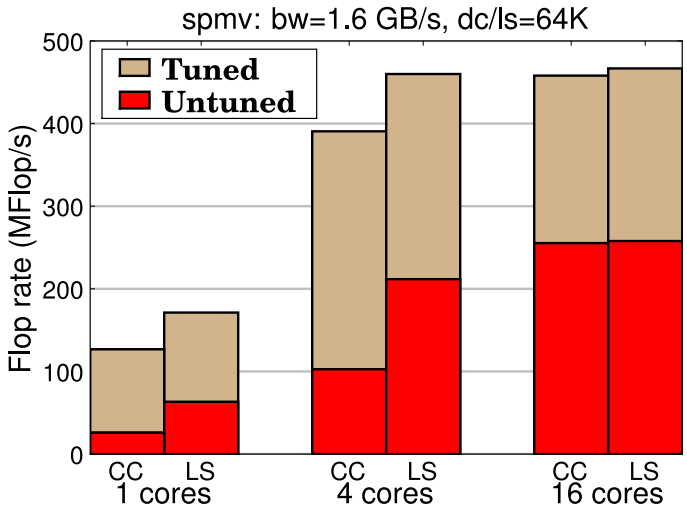
GEMM



Stencil



SpMV

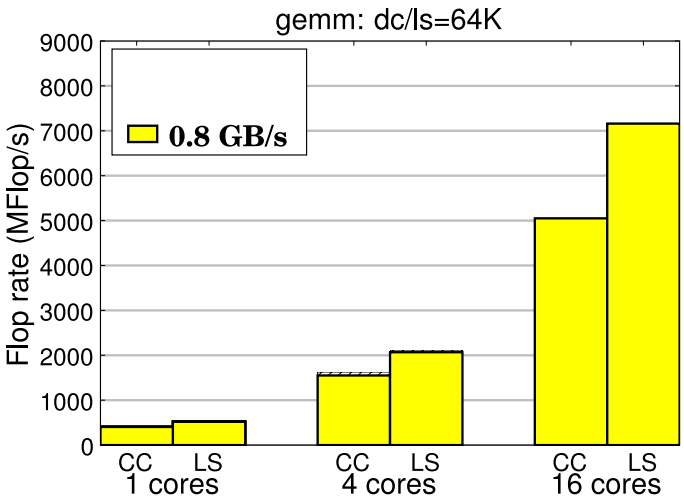


- GEMM gains a lot from tuning
- Software-managed caches get better performance
- Bandwidth-saturation for stencil and SpMV

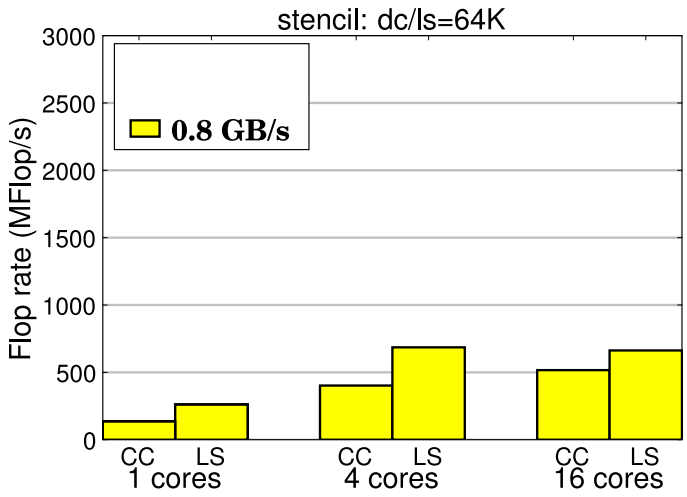
Effect of memory bandwidth on tuned performance

D-cache/local store = 64 KB (CC=cache, LS=local store)

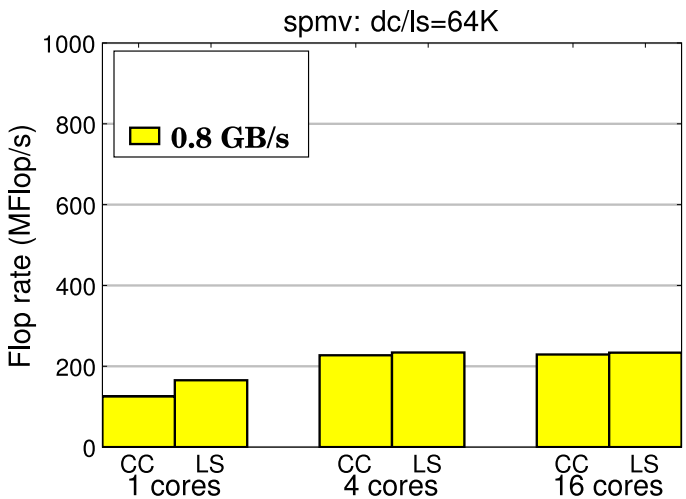
GEMM



Stencil



SpMV

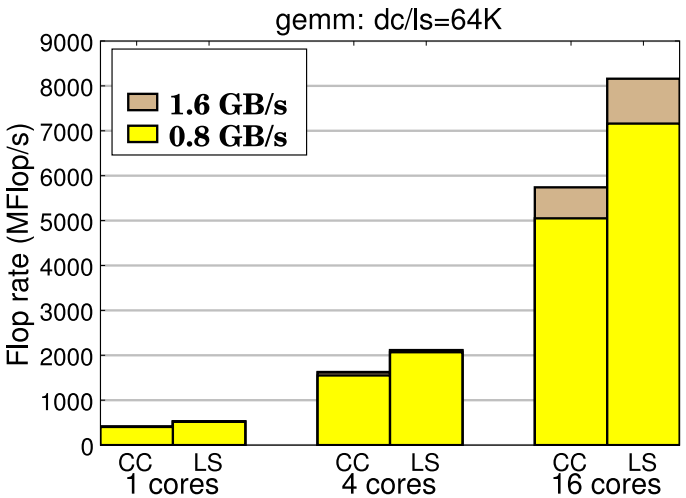


- GEMM least sensitive to memory bandwidth
- SpMV performance scales with memory bandwidth for enough cores

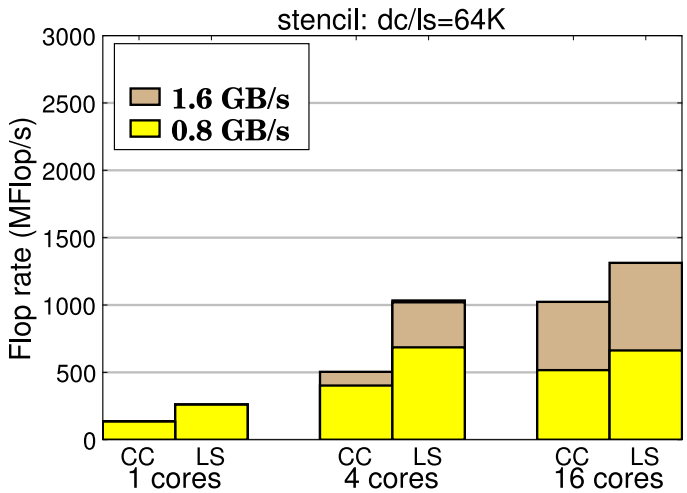
Effect of memory bandwidth on tuned performance

D-cache/local store = 64 KB (CC=cache, LS=local store)

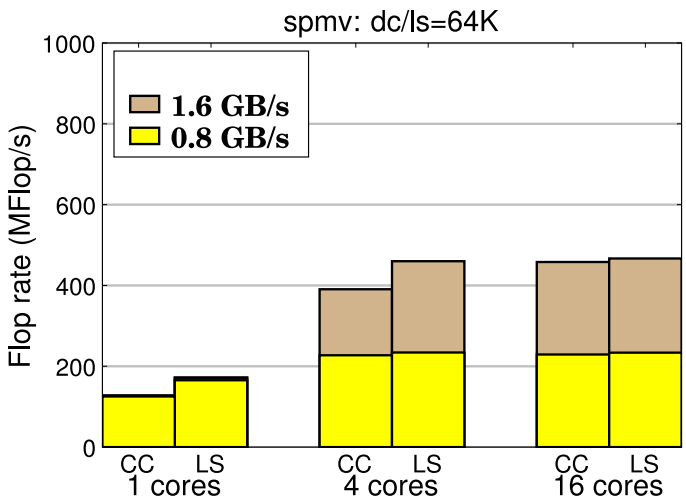
GEMM



Stencil



SpMV

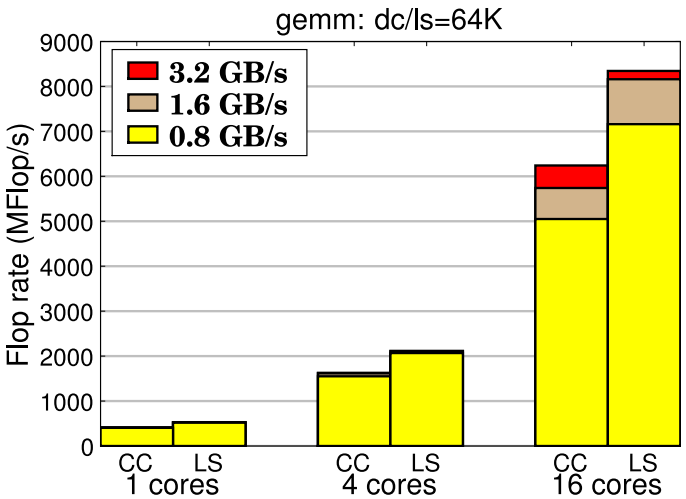


- GEMM least sensitive to memory bandwidth
- SpMV performance scales with memory bandwidth for enough cores

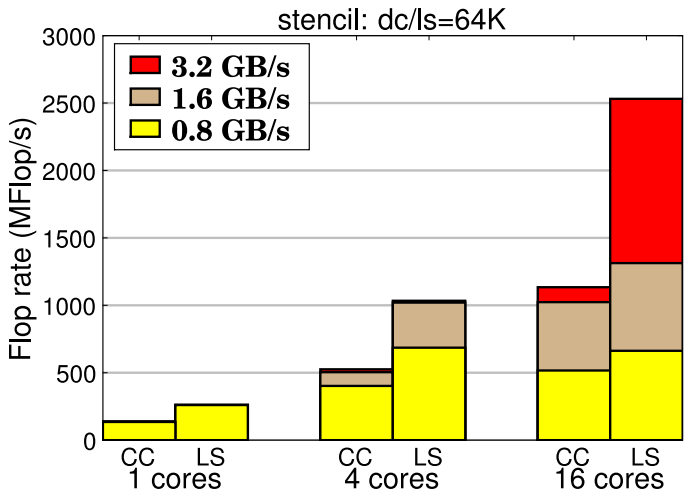
Effect of memory bandwidth on tuned performance

D-cache/local store = 64 KB (CC=cache, LS=local store)

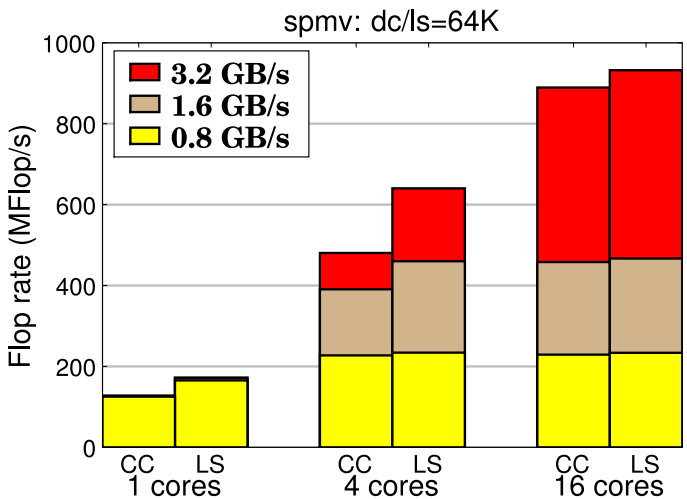
GEMM



Stencil



SpMV

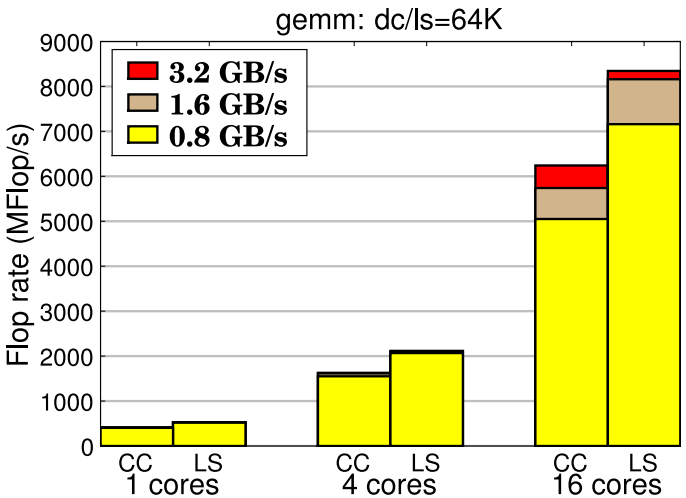


- GEMM least sensitive to memory bandwidth
- SpMV performance scales with memory bandwidth for enough cores

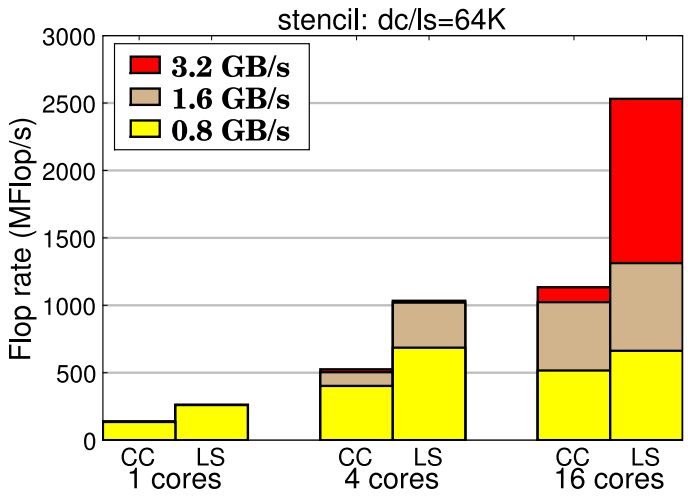
Effect of memory bandwidth on tuned performance

D-cache/local store = 64 KB (CC=cache, LS=local store)

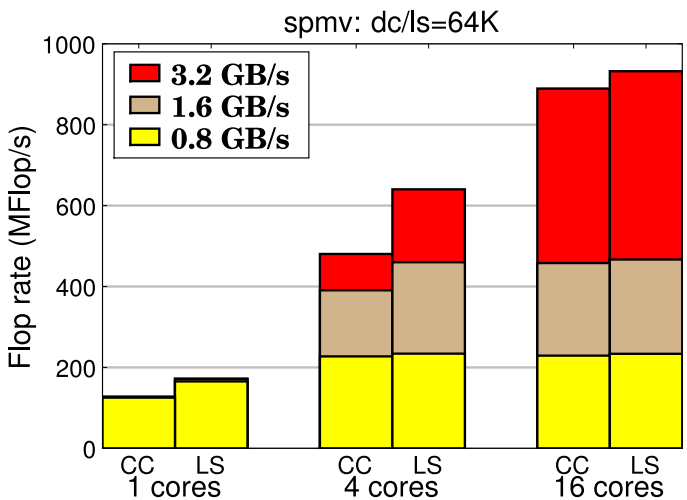
GEMM



Stencil

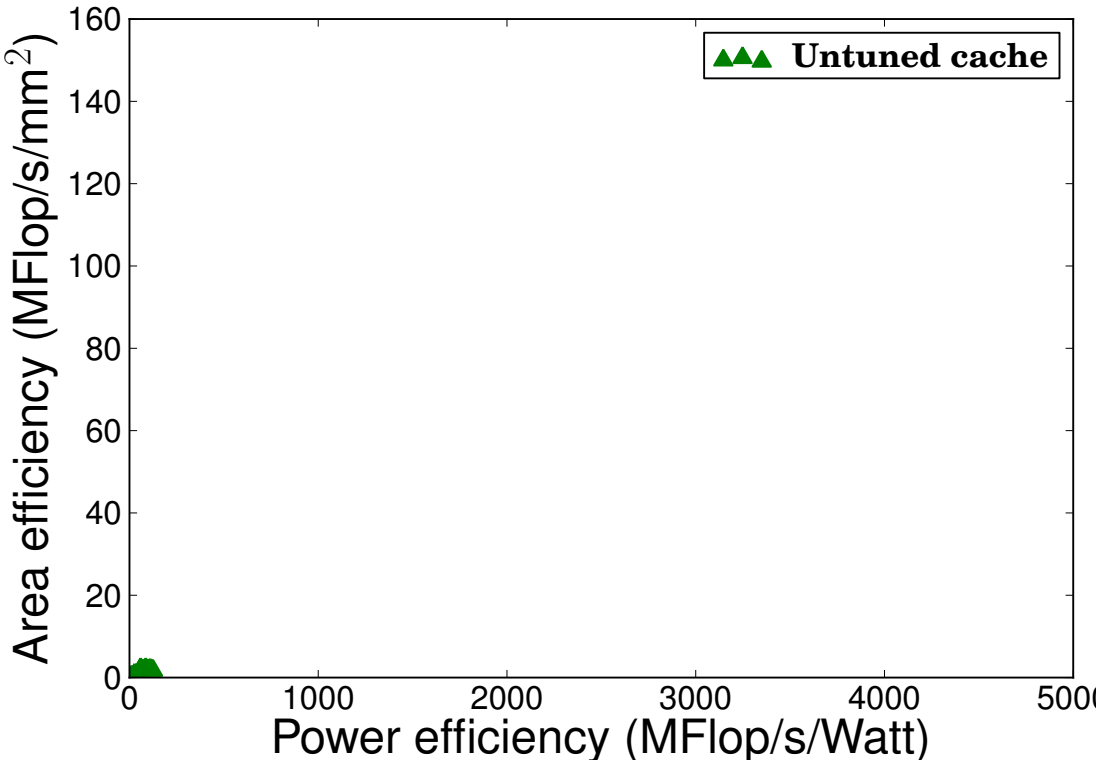


SpMV



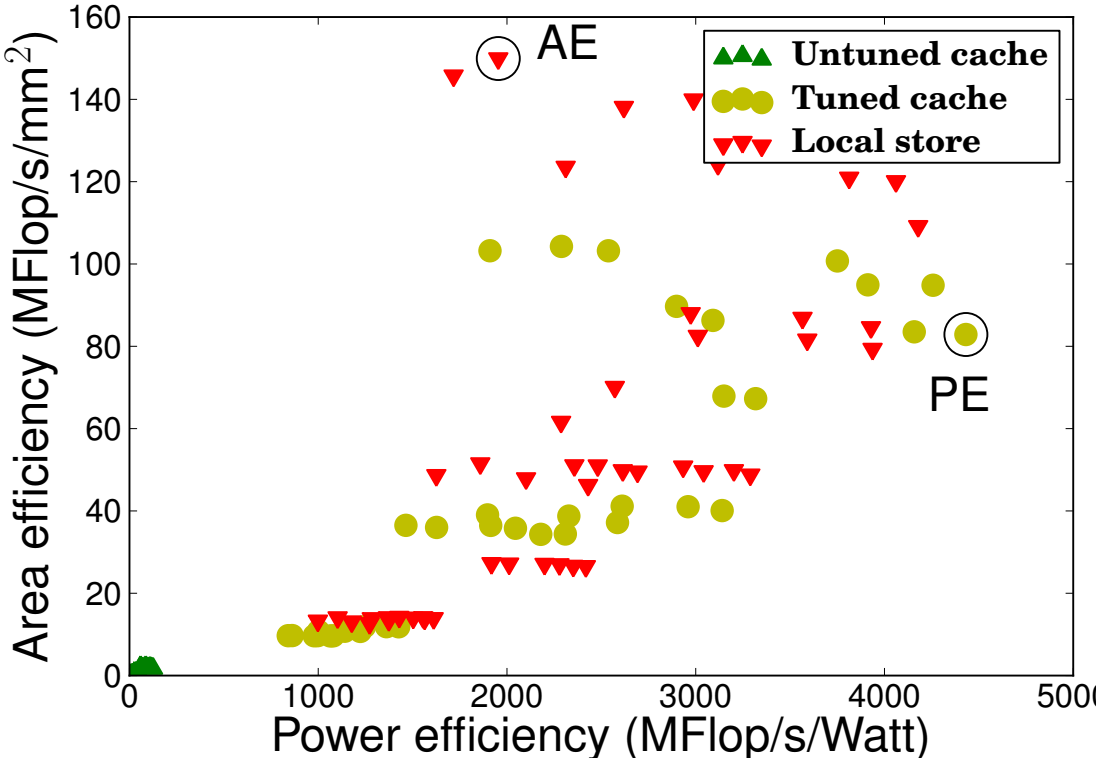
- GEMM least sensitive to memory bandwidth
- SpMV performance scales with memory bandwidth for enough cores

Efficiency Improvements: GEMM



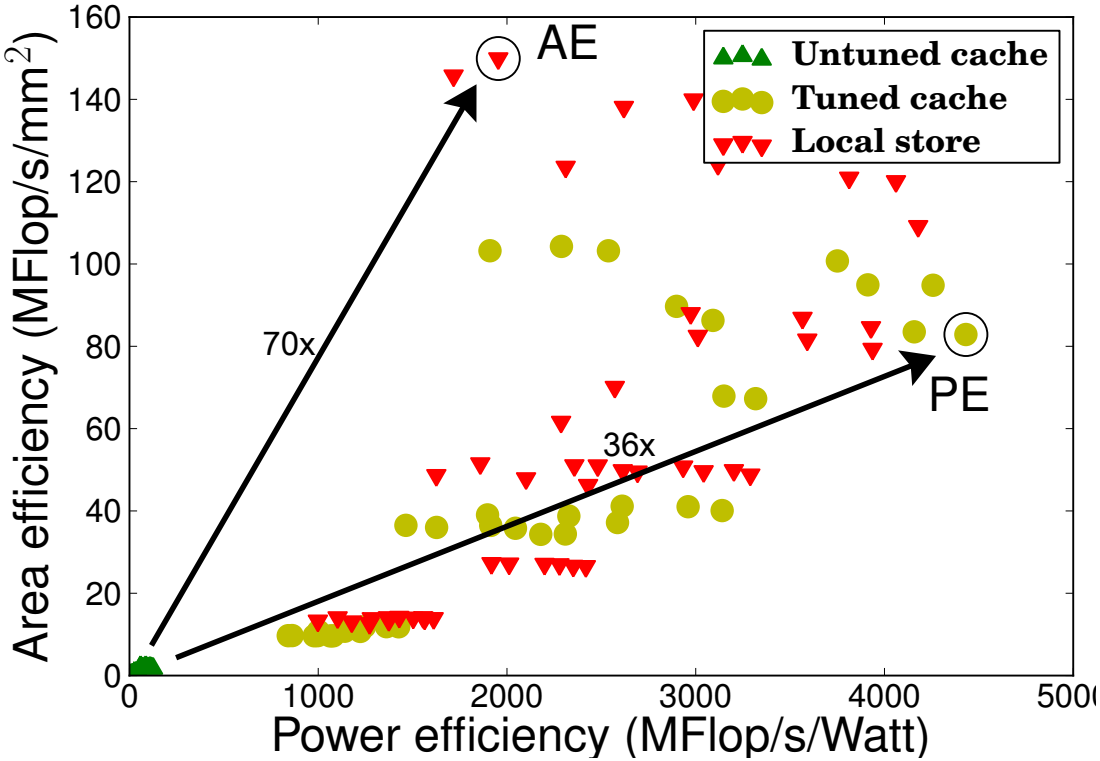
- Each point represent a HW config (AE = most area efficient, PE = most power efficient)
  - Best SW performance chosen by autotuner used for computing efficiencies
- Efficiency improvements from SW tuning dramatic

Efficiency Improvements: GEMM



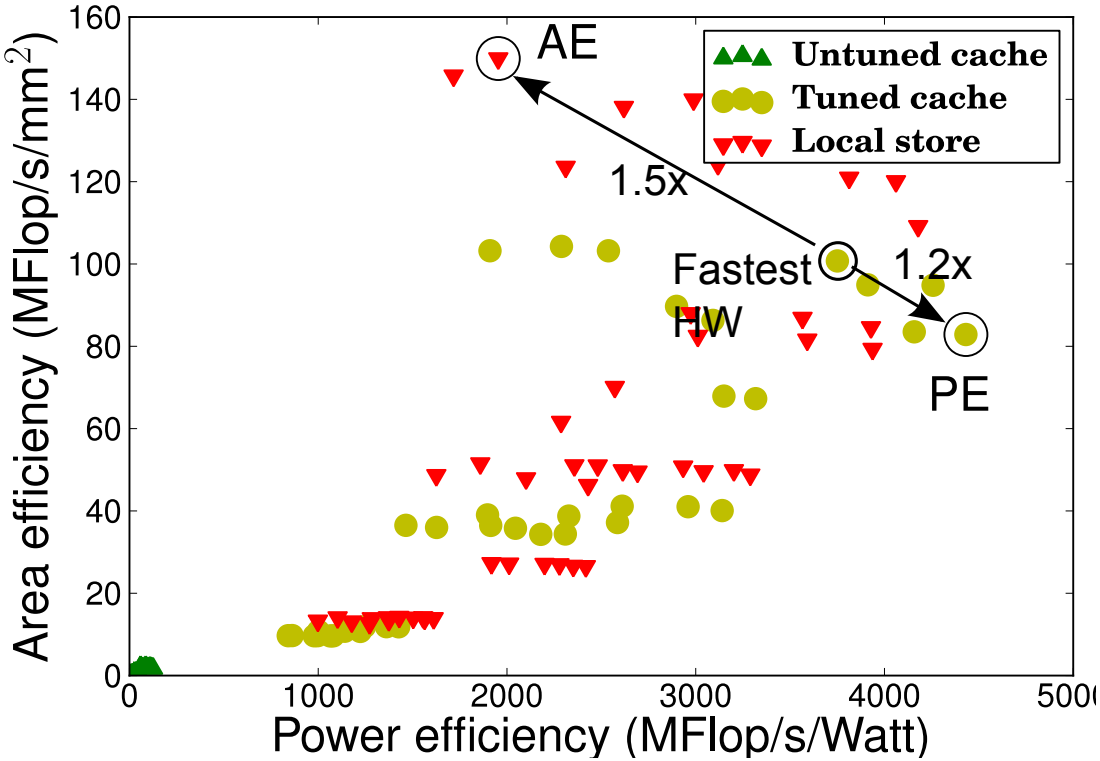
- Each point represent a HW config (AE = most area efficient, PE = most power efficient)
  - Best SW performance chosen by autotuner used for computing efficiencies
- Efficiency improvements from SW tuning dramatic

Efficiency Improvements: GEMM



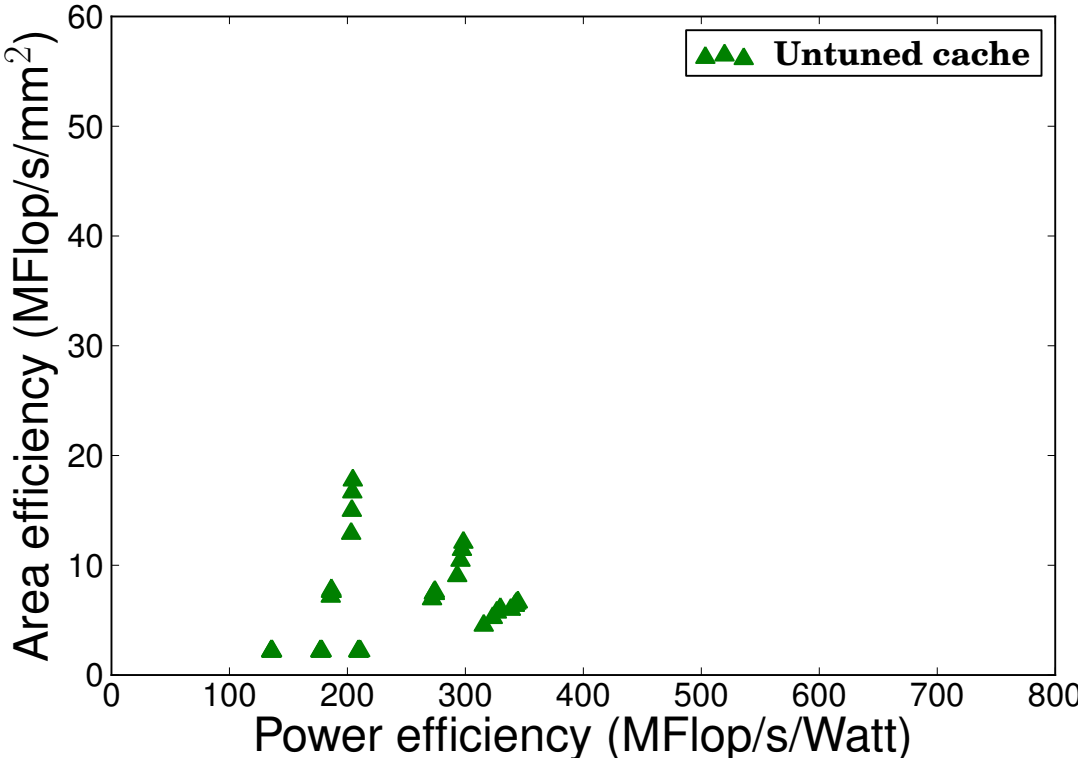
- Each point represent a HW config (AE = most area efficient, PE = most power efficient)
  - Best SW performance chosen by autotuner used for computing efficiencies
- Efficiency improvements from SW tuning dramatic

Efficiency Improvements: GEMM



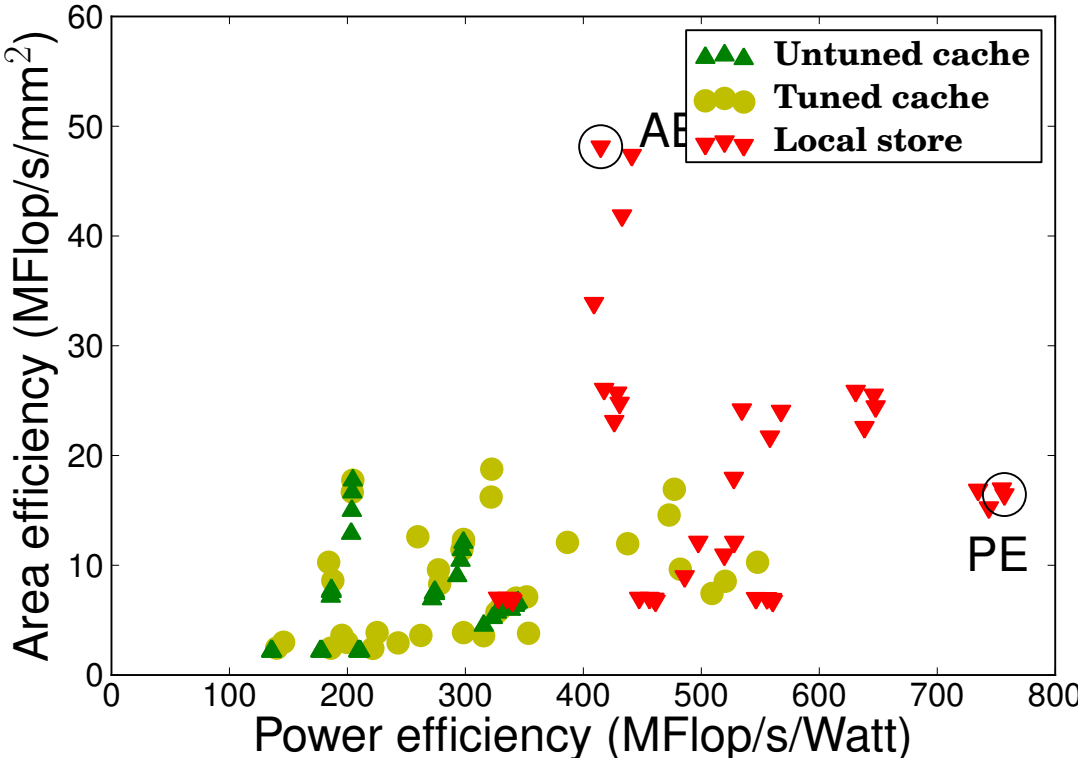
- Each point represent a HW config (AE = most area efficient, PE = most power efficient)
  - Best SW performance chosen by autotuner used for computing efficiencies
- Efficiency improvements from SW tuning dramatic

Efficiency Improvements: Stencil



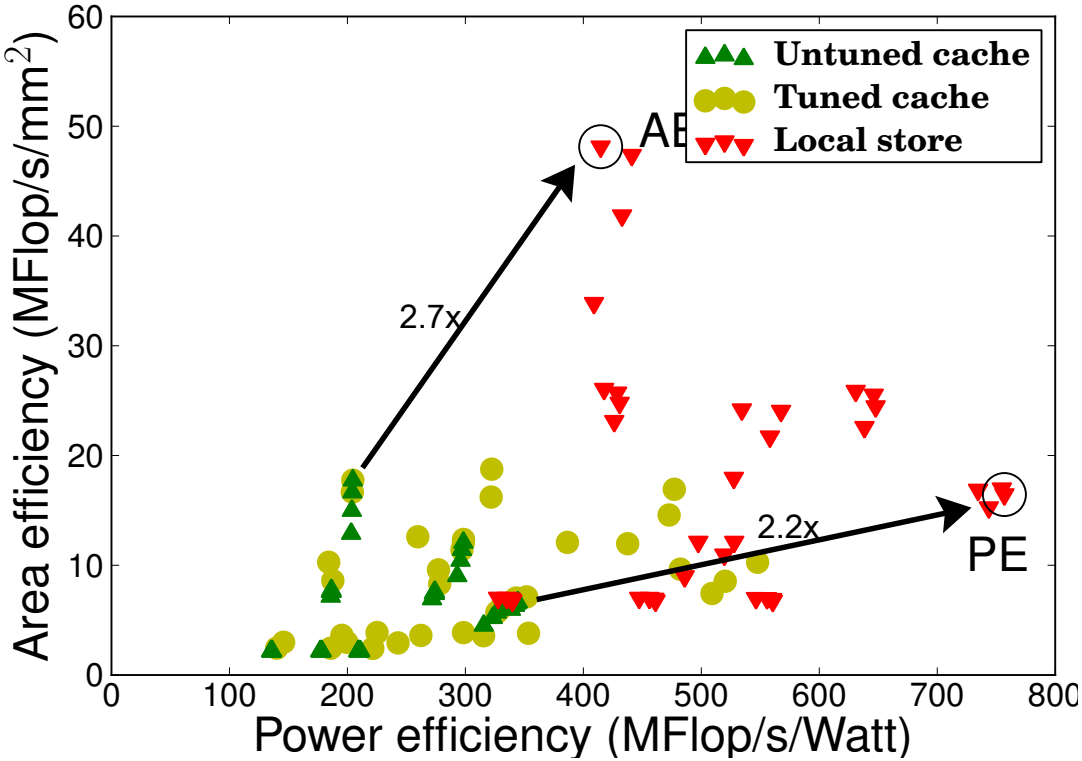
- Each point represent a HW config
  - Best SW performance chosen by autotuner used for computing efficiencies

Efficiency Improvements: Stencil



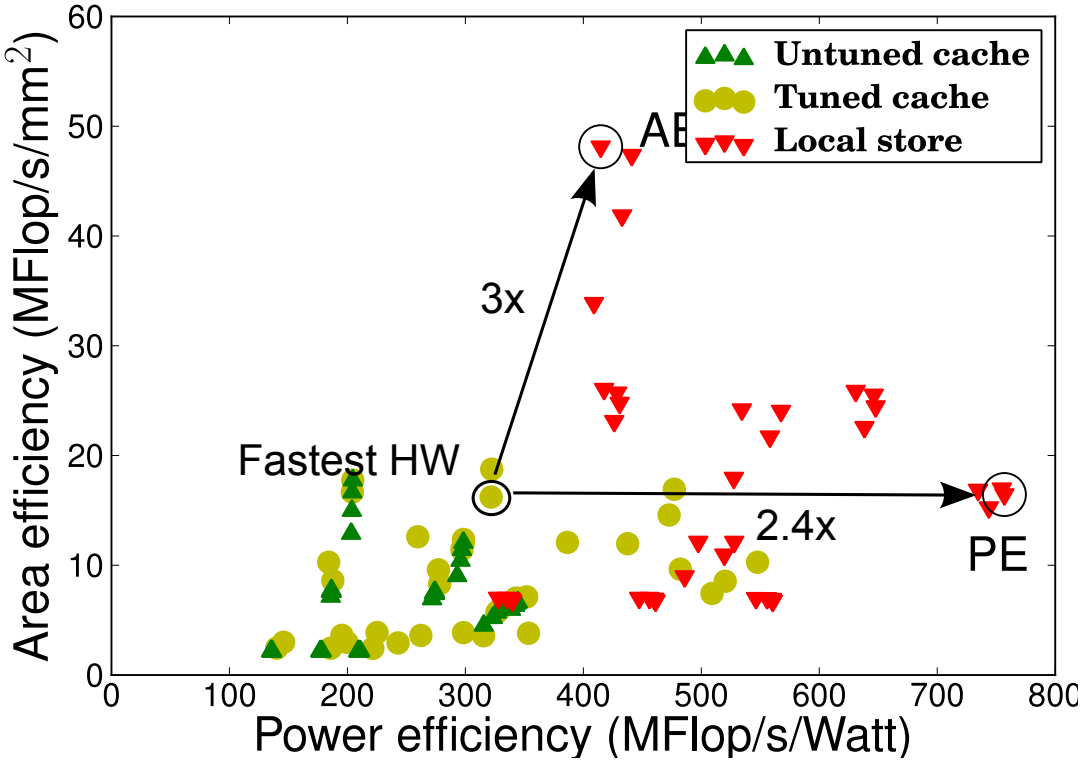
- Each point represent a HW config
  - Best SW performance chosen by autotuner used for computing efficiencies

Efficiency Improvements: Stencil



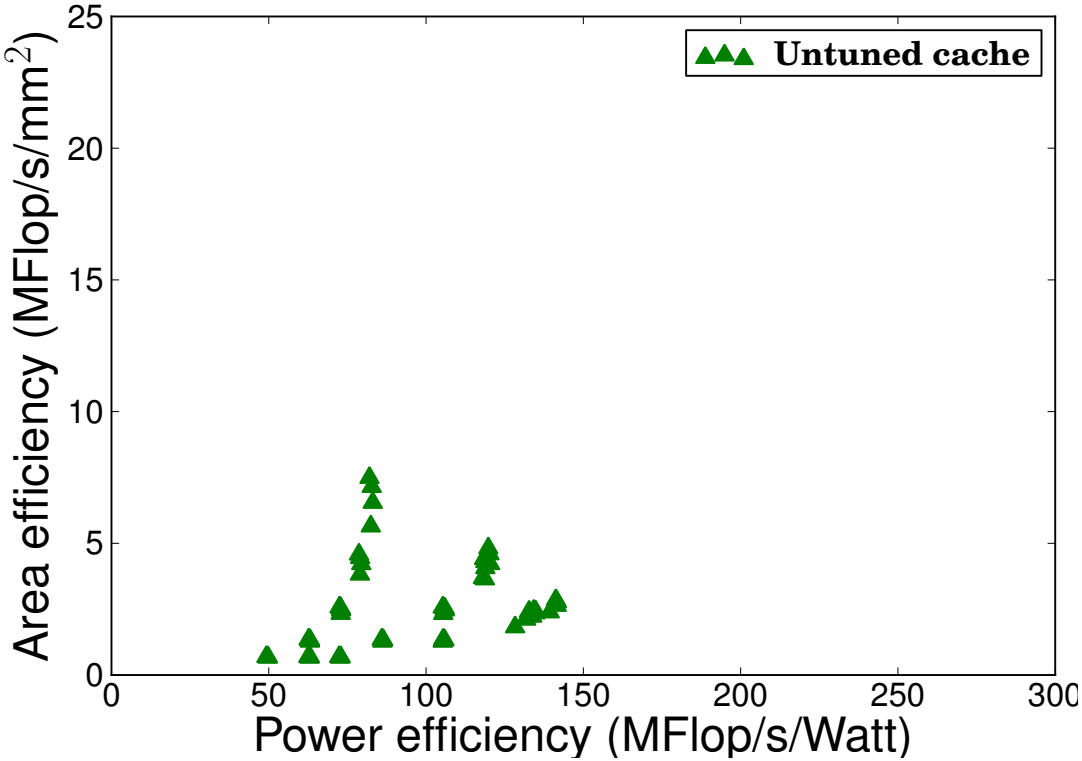
- Each point represent a HW config
  - Best SW performance chosen by autotuner used for computing efficiencies

Efficiency Improvements: Stencil



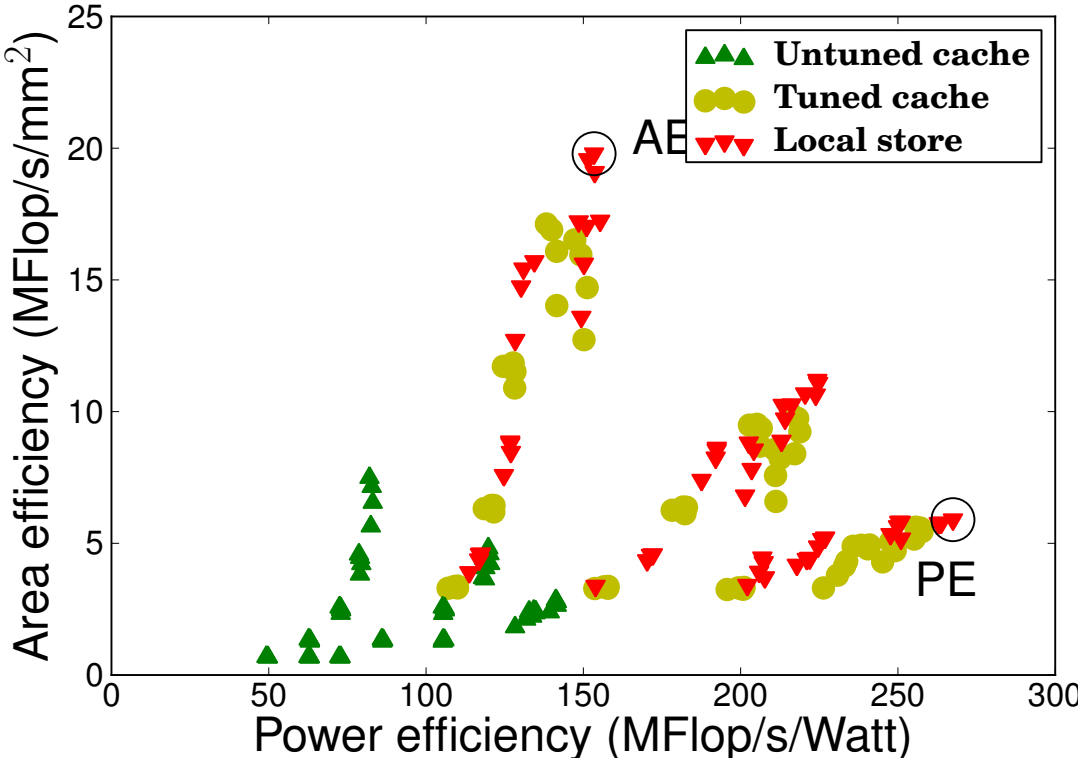
- Each point represent a HW config
  - Best SW performance chosen by autotuner used for computing efficiencies

Efficiency Improvements: SpMV



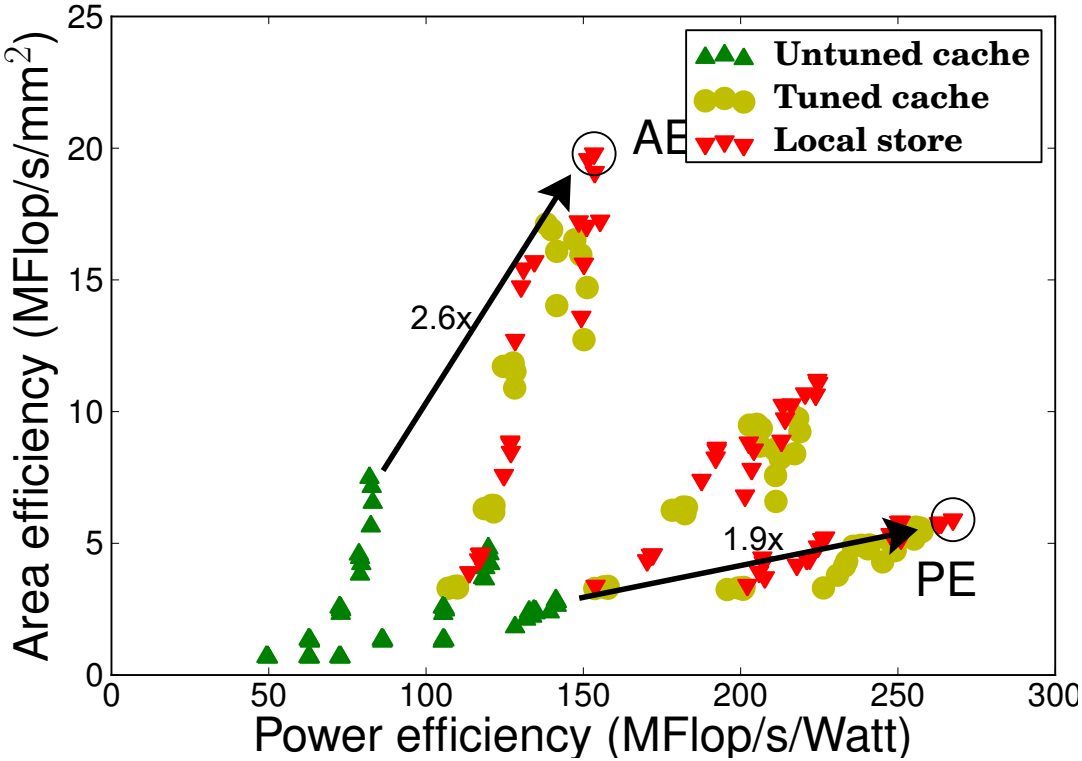
- Each point represent a HW config
  - Best SW performance chosen by autotuner used for computing efficiencies

Efficiency Improvements: SpMV



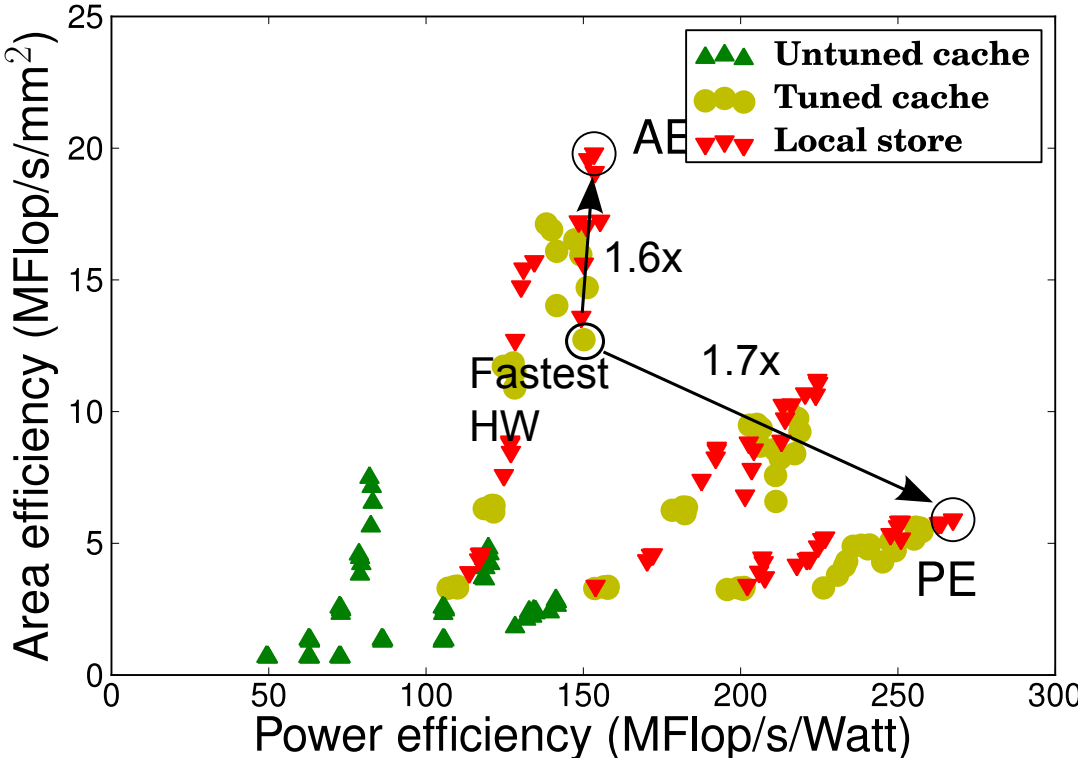
- Each point represent a HW config
  - Best SW performance chosen by autotuner used for computing efficiencies

Efficiency Improvements: SpMV



- Each point represent a HW config
  - Best SW performance chosen by autotuner used for computing efficiencies

Efficiency Improvements: SpMV



- Each point represent a HW config
  - Best SW performance chosen by autotuner used for computing efficiencies

Co-tuning for multiple kernels

- Results so far find best HW config given kernel
- How about an application composed of multiple kernels?
- Simple case: kernels dont interact, all flops contributed by the given kernels
  $\Rightarrow$ sufficient to tune kernels instead of full application
  - Performance/power for application on a HW config = weighted performance/power of kernels on the config
  - Weights = relative contribution of different kernels

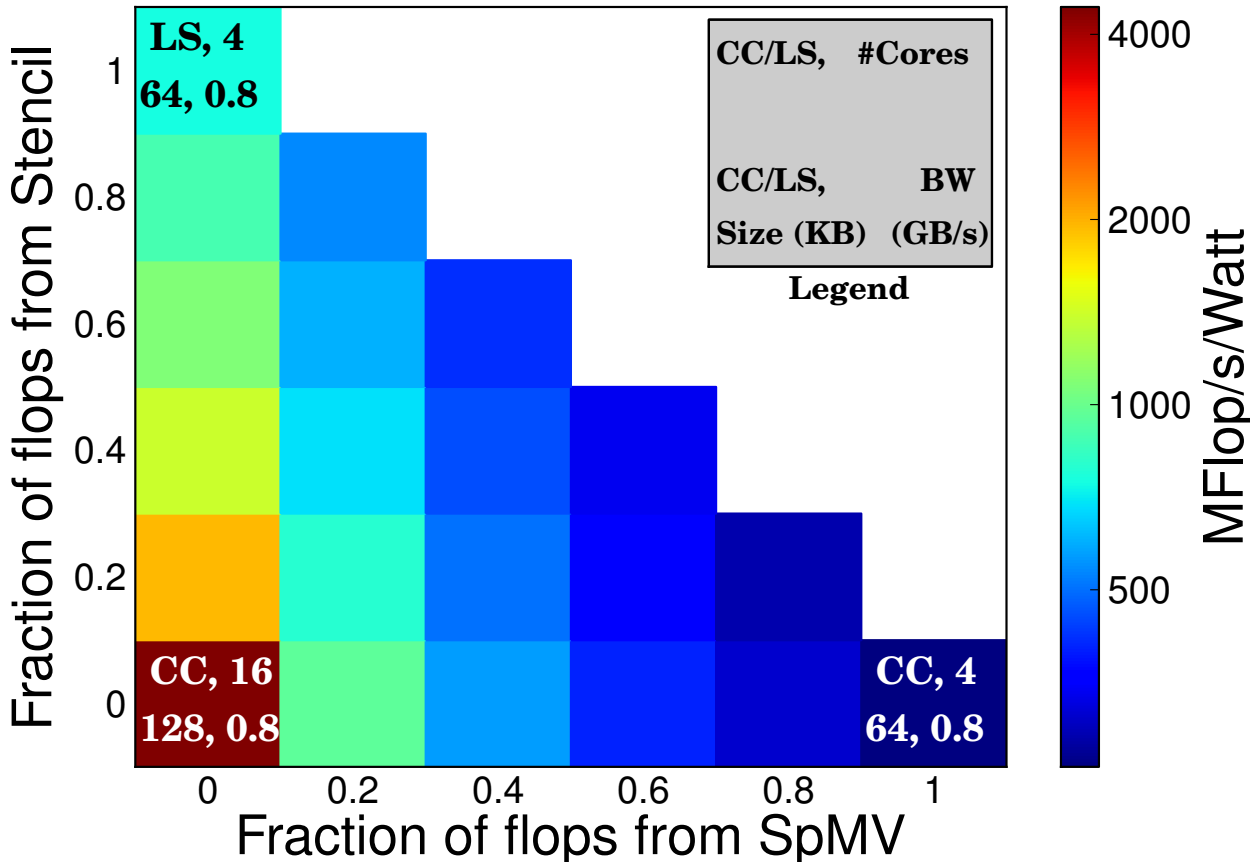
Co-tuning for multiple kernels

- Results so far find best HW config given kernel
- How about an application composed of multiple kernels?
- Simple case: kernels dont interact, all flops contributed by the given kernels
  $\Rightarrow$ sufficient to tune kernels instead of full application
  - Performance/power for application on a HW config = weighted performance/power of kernels on the config
  - Weights = relative contribution of different kernels

Co-tuning for multiple kernels

- Results so far find best HW config given kernel
- How about an application composed of multiple kernels?
- Simple case: kernels dont interact, all flops contributed by the given kernels
  $\Rightarrow$ sufficient to tune kernels instead of full application
  - Performance/power for application on a HW config = weighted performance/power of kernels on the config
  - Weights = relative contribution of different kernels

Tuning Multi-Kernel Application



Each box represents the most power-efficient HW config for the given relative weights of kernels

Summary of results

- Baseline: SW tuning done on the fastest HW config
- GEMM: 1.2× and 1.5× improvements in power and area efficiencies
- Stencil: 2.4× and 3× improvements in power and area efficiencies
- SpMV: 1.7× and 1.6× improvements in power and area efficiencies
- Weighted combination of GEMM, stencil, SpMV: improvements vary from 1.2× to 2.4× depending on relative contribution

Outline

Conclusions and Future Work

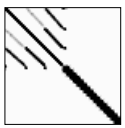
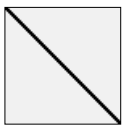
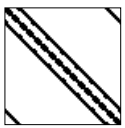
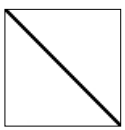
- Novel approach to designing power-efficient supercomputers
  - Leverage software auto-tuning to improve efficiency
  - Power efficiency improved 1.2–2.4×,
    area efficiency improved 1.5–3×
  - Improvements also in multi-kernel applications
  - Co-tuning can cut down procurement and running costs
- Future work
  - Explore a larger HW design space
    ⇒ need intelligent exploration
  - Use FPGA-based emulation of hardware for speeding up exploration
  - Efficiently co-tuning for applications with interacting kernels
  - Green Flash design

Conclusions and Future Work

- Novel approach to designing power-efficient supercomputers
  - Leverage software auto-tuning to improve efficiency
  - Power efficiency improved 1.2–2.4×,
    area efficiency improved 1.5–3×
  - Improvements also in multi-kernel applications
  - Co-tuning can cut down procurement and running costs
- Future work
  - Explore a larger HW design space
    ⇒ need intelligent exploration
  - Use FPGA-based emulation of hardware for speeding up exploration
  - Efficiently co-tuning for applications with interacting kernels
  - Green Flash design

Questions?

Kernel 3: Matrices

spyplot	Name	Dimensions	Nonzeros (nnz/row)	Description
	Dense	2K x 2K	4.0M (2K)	Dense matrix in sparse format
	FEM / Spheres	83K x 83K	6.0M (72)	FEM concentric spheres
	FEM / Cantilever	62K x 62K	4.0M (65)	FEM cantilever
	Wind Tunnel	218K x 218K	11.6M (53)	Pressurized wind tunnel
	QCD	49K x 49K	1.90M (39)	Quark propagators (QCD/LGT)
	FEM/Ship	141K x 141K	3.98M (28)	FEM Ship section/detail
	Epidemiology	526K x 526K	2.1M (4)	2D Markov model of epidemic
	Circuit	171K x 171K	959K (6)	Motorola circuit simulation

- SpMV performance dependent on matrix nonzero pattern
- Matrices chosen to represent different applications
- Dense matrix in sparse format used for tuning
- For each HW config, SpMV performance = performance of median matrix