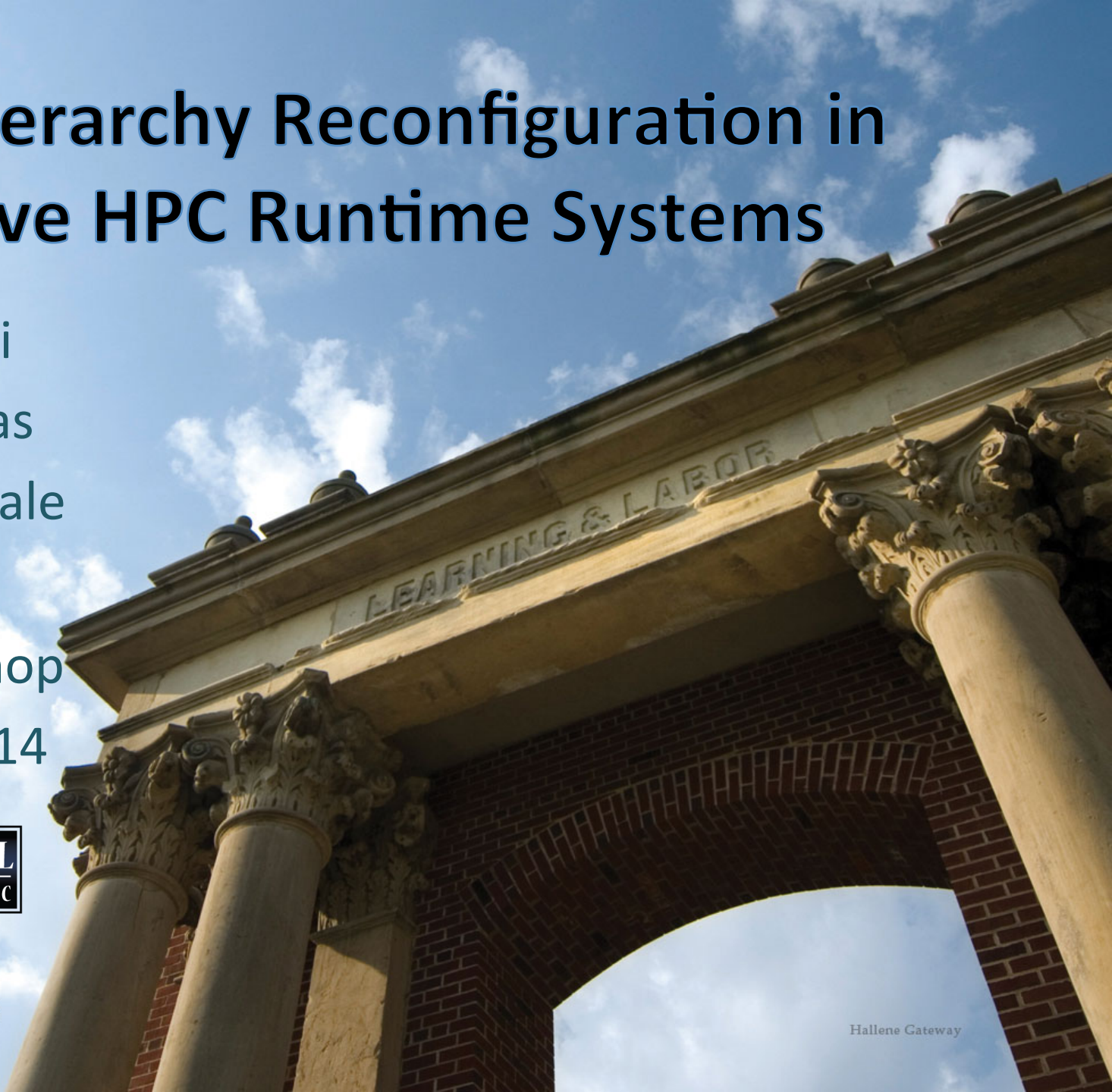
Cache Hierarchy Reconfiguration in Adaptive HPC Runtime Systems

Ehsan Totoni
Josep Torrellas
Laxmikant V. Kale

Charm Workshop
April 29th, 2014

PARALLEL PROGRAMMING LAB
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS
PPL UIUC

ILLINOIS



Exascale Power Challenge

- Tianhe-2
- ~34 PFlop/s Linpack
- ~18 MW power
- Goal: ExaFlop/s at 20MW
- ~26 times more energy efficiency needed

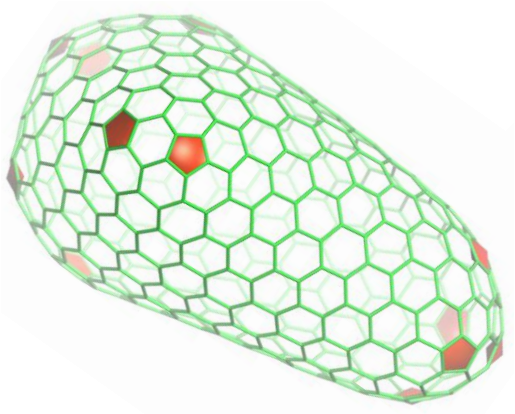
Top500.org November 2013

Caches

- Caches consume a large fraction of processor's power
  - 40% in POWER7, after many techniques
- Getting larger every day
  - Intel Xeon E7-88702: 30MB of SRAM L3
  - IBM POWER8: 96MB of eDRAM L3
- Fixed design, but applications are different
  - E.g. potentially no locality in pointer chasing "Big Data"

Cache Energy Waste

- Scenario: NAMD on Blue Waters
  - HIV simulations, only 64 million atoms
    - 48 bytes atom state (position & velocity)
    - Some transient data (multicasts)
    - Assuming 400 bytes/atom, 25.6 GB
  - 4000 Cray-XE nodes
    - 32 MB of L2 and 32 MB L3 each -> 256 GB of cache!
    - 90% of capacity not unused
    - (there is nothing wrong with NAMD!)
  - 16 days wall clock time, not best use of caches.. Huge waste!



Cache Reconfiguration

- Turning off cache ways to save energy proposed
- Two main issues:
  - Predicting the applications future
  - Finding the best cache hierarchy configuration
- We solve both on HPC systems

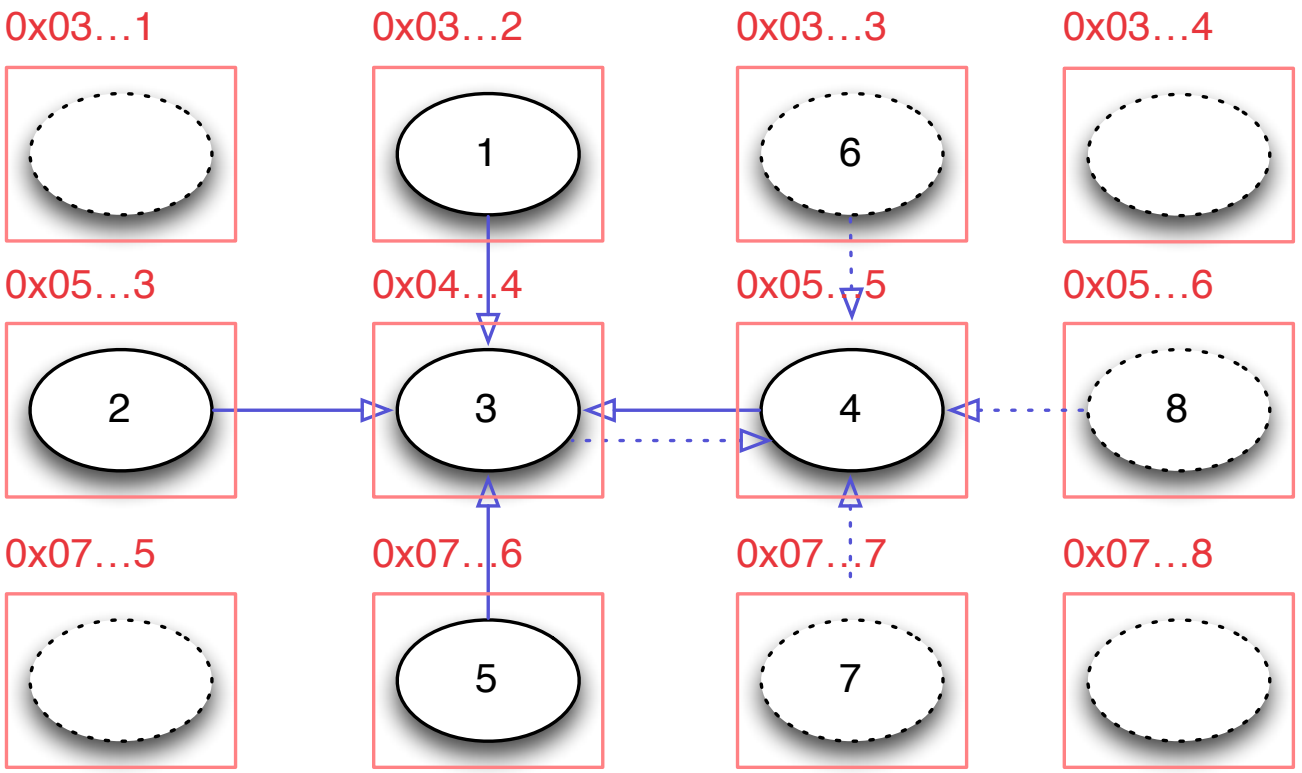
HPC Systems

- Many processors are commodity
  - Not necessarily designed for HPC
- Provisioning different than non-HPC
  - No multi-programming, time-sharing, co-location
  - Large, long jobs
  - High Predictability

HPC Applications

- Properties of algorithms in common HPC apps:
  - Particle interactions (MiniMD and CoMD)
    - Force computation of entities
    - Small domain, high temporal locality
  - Stencil computations (CloverLeaf and MiniGhost)
    - Update of grid points with stencils
    - Large domain, low temporal locality
  - Sparse Linear Algebra (HPCCG, MiniFE, and MiniXyce)
    - Update of grid points with SpMV
    - Often large domain, low temporal locality

Stencil Access Pattern



Adaptive RTS Approach

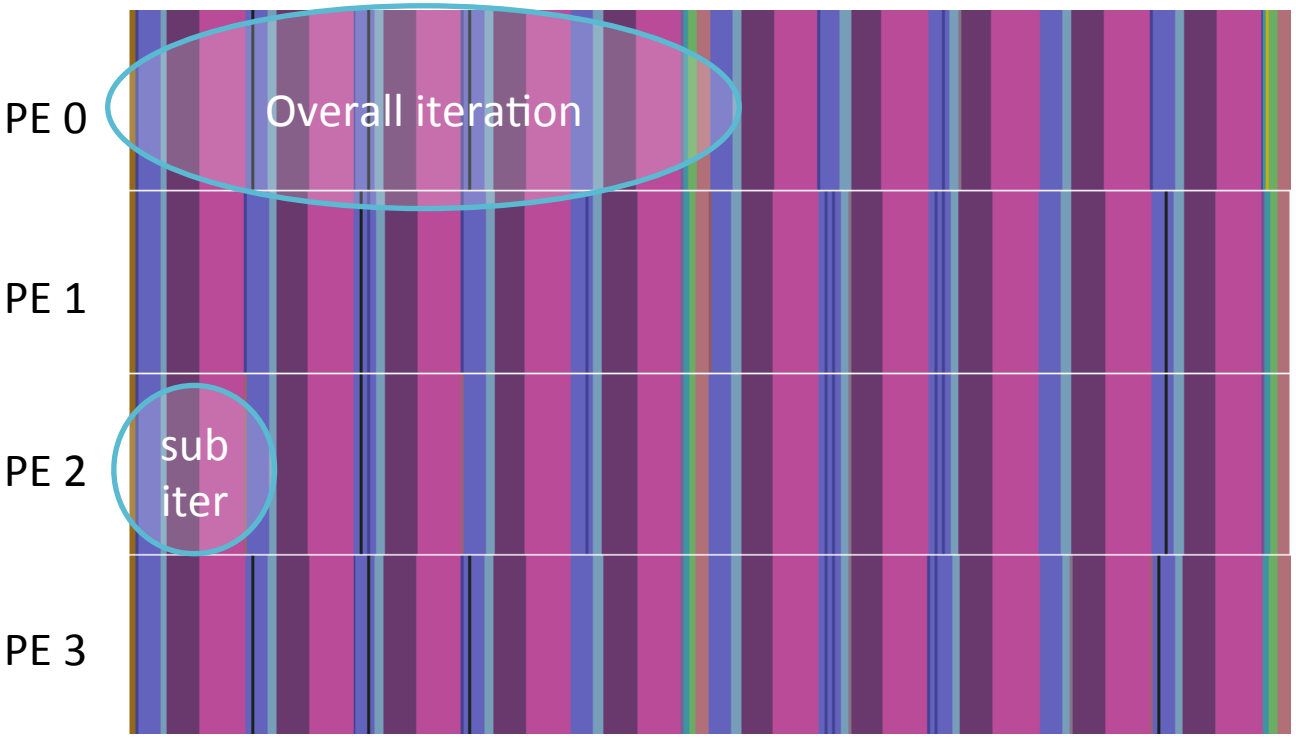
- HPC applications are iterative
  - *Persistence*: Same pattern repeats
  - RTS can monitor application, predict future
- Single Program Multiple Data (SPMD)
  - Different processors doing the same thing
  - RTS can try cache configurations exhaustively
- RTS can apply best cache configuration
  - Monitor, re-evaluate regularly

Reconfiguration Units

- RTS tracks Sequential Execution Blocks (SEBs)
  - Computations between communication calls
- Identified by characteristic information
  - Communication calls and their arguments
  - Duration
  - Key performance counters
- Usually repeated in every iteration

MILC's Pattern

- Hierarchical iteration structure

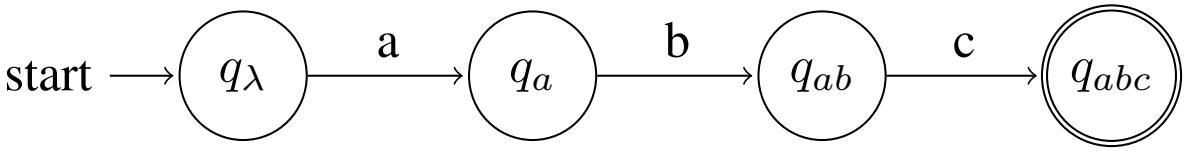


Identifying Iteration Structure

- RTS needs to identify iterative structure
  - Difficult in most general sense
- Using Formal Language Theory
  - Define each SEB as a symbol of an alphabet Σ
  - An iterative structure is a *regular language*
    - Easy to prove by construction
  - Each execution is a *word*

Pattern Recognition

- In profiling, RTS sees a stream of SEBs (symbols)
  - Needs to recognize the pattern
  - *Learning a regular language from text*
  - Build a Deterministic Finite Automaton (DFA)
- Prefix Tree Acceptor (PTA)
  - A state for each prefix
  - Not too large in our application



Evaluation Methodology

- Mantevo mini-app suite
  - Representative inputs
  - Assume MPI+OpenMP
  - Identify unique SEBs
- SESC cycle-accurate simulator
  - Simulate different configurations for each SEB
- Model cache power/energy using CACTI

Evaluation Results

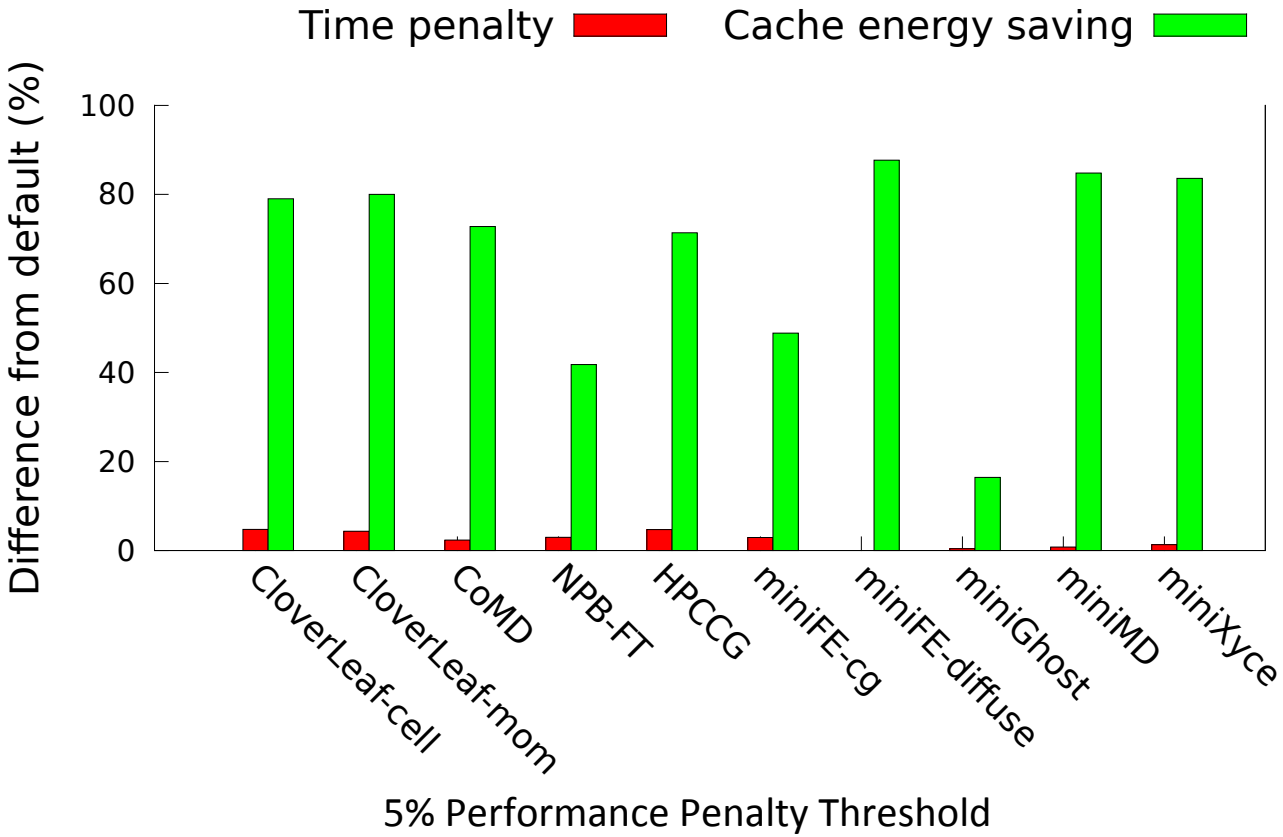
Best configuration depends on:

- Application type
- Input size

Format: <ways turned on>/<total number of ways>

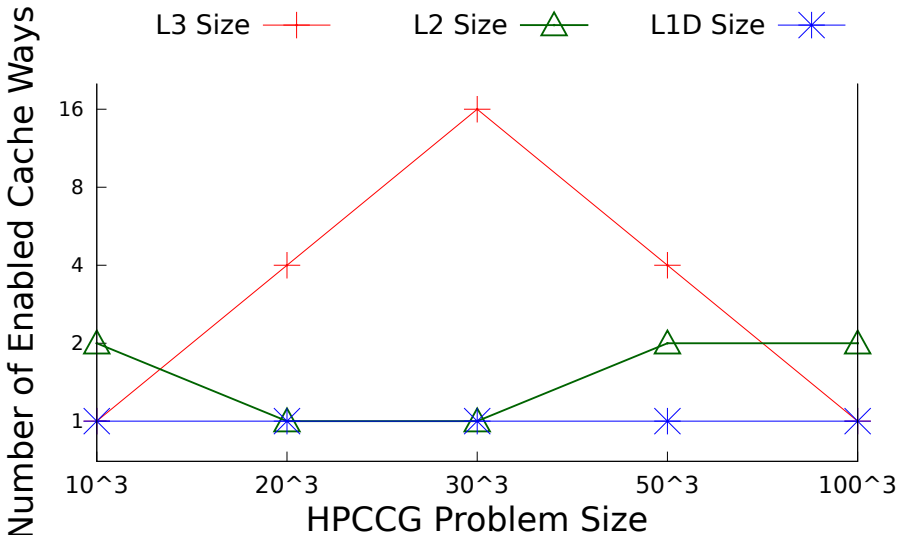
Mini-App	L1D	L1I	L2	L3
CloverLeaf-cell	1/4	1/2	2/8	16/16
CloverLeaf-mom	1/4	1/2	2/8	16/16
CoMD	1/4	1/2	2/8	8/16
NPB-FT	1/4	2/2	4/8	16/16
HPCCG	1/4	1/2	2/8	16/16
miniFE-cg	1/4	1/2	2/8	16/16
miniFE-diffuse	1/4	1/2	1/8	1/16
miniGhost	1/4	1/2	2/8	16/16
miniMD	2/4	1/2	2/8	1/16
miniXyce	1/4	1/2	4/8	1/16

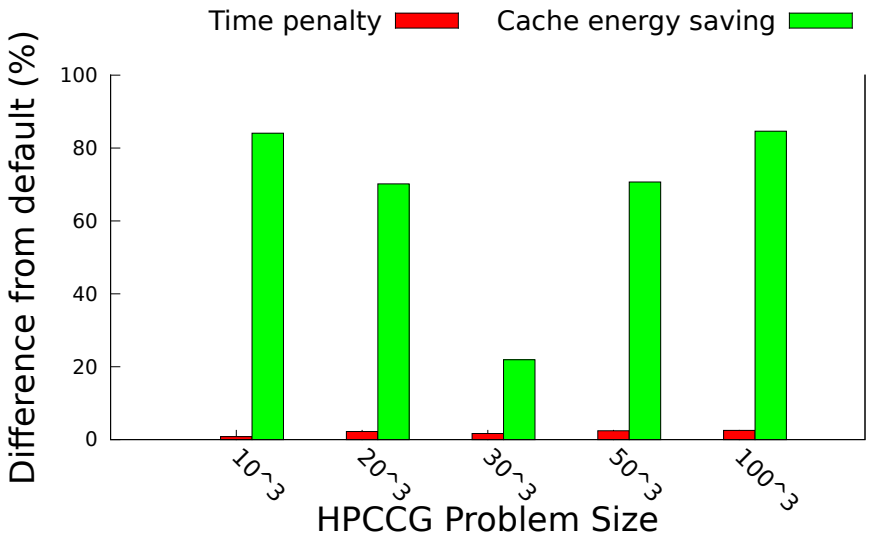
Evaluation Results



67% cache energy saving (28% in processor) on average

Adapt to Problem Size





Adapting to problem size is crucial.

Reconfigurable Streaming

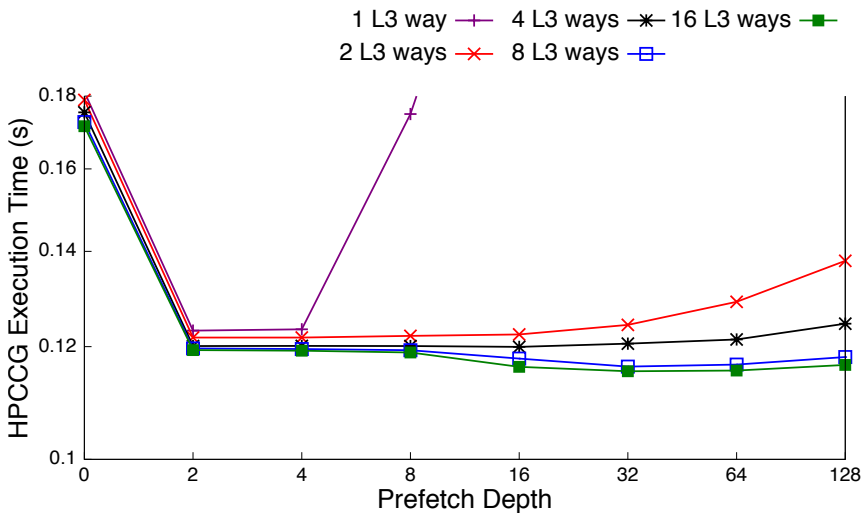
- Streaming: predict data and prefetch for simple memory access patterns
  - Two important parameters:
  - Cache size to use
  - Prefetch depth
- Can waste energy and memory bandwidth
  - Too deep/small cache evicts useful data
  - Prefetch enough data to hide memory latency

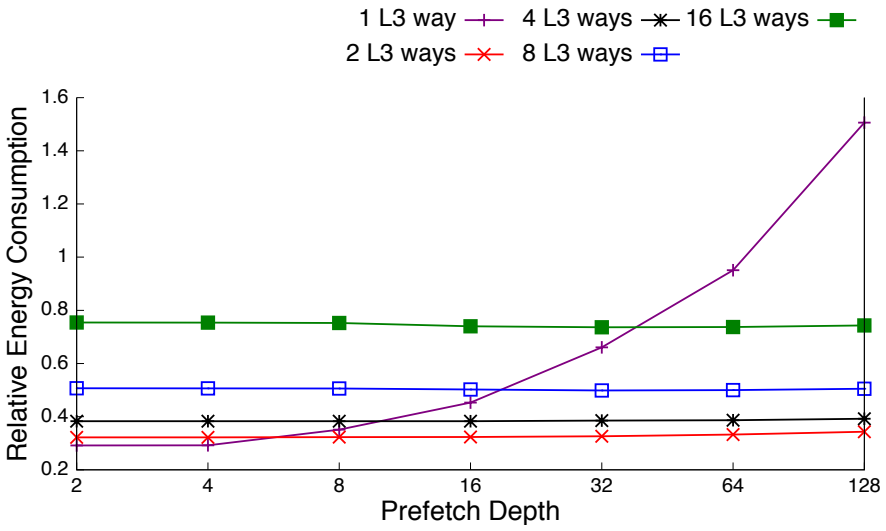
Reconfigurable Streaming

- RTS can tune cache size and depth
  - Similar to previous discussion
- Hardware implementation:
  - Prefetcher has an adder to generate next address
  - One input can be controlled by RTS as a system register
  - Does not have overheads of repetitive prefetch instructions

Performance/Energy Tradeoff

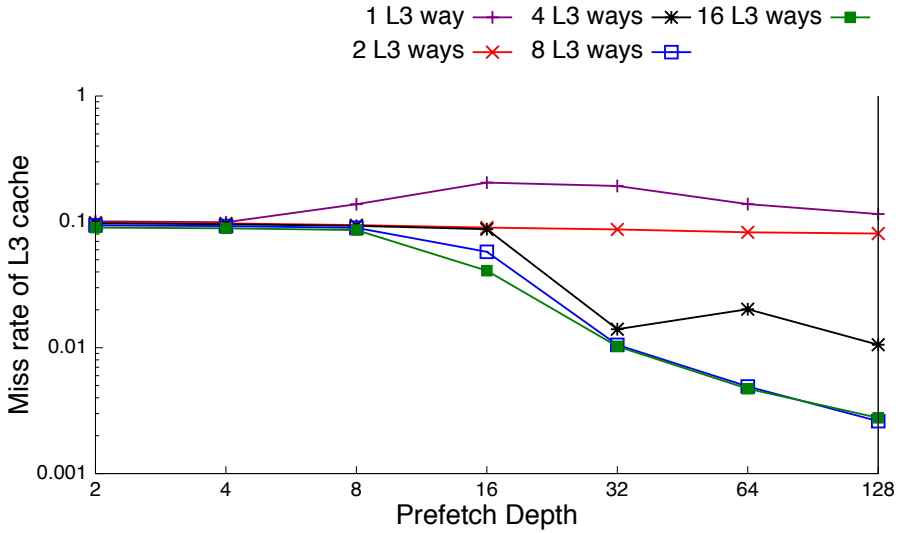
- Small gains in performance might have high energy cost

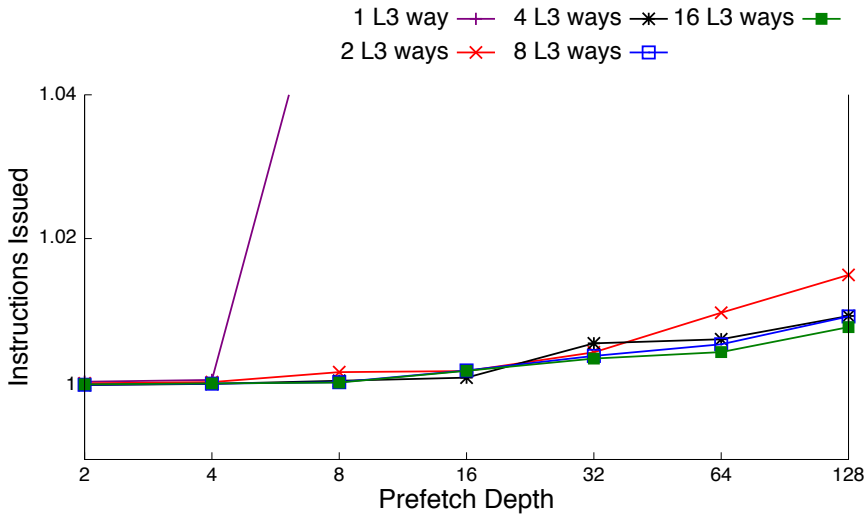




Hardware Complexities

- Wrong speculative path is accelerated with deeper prefetch
- Intervenes with useful computation





Related Work

- Automatic cache hierarchy reconfiguration in hardware had been explored extensively
  - Survey by Zang and Gordon-Ross
  - Hardware complexity -> energy overhead
  - Hard to predict application behavior in hardware
    - Small "window"
    - Choosing best configuration
- Compiler directed cache reconfiguration (Hu et al.)
  - Compiler's analysis is usually limited
    - Many assumptions for footprint analysis
      - Simple affine nested loops
      - Simple array indices (affine functions of constants and index variables)
    - Not feasible for real applications

PPL UIUC

Conclusion

- Caches consume a lot of energy (40%>)
- Adaptive RTS can predict application's future
  - Using Formal Language Theory
- Best cache configuration can be found in parallel (SPMD model)
  - 67% of cache energy is saved on average
- Reconfigurable streaming
  - Improves performance and saves energy
  - 30% performance and 75% energy in some cases

PPL UIUC

Future Work

- Prototype machine (MIT Angstrom?) and runtime (Charm++ PICS)
- Find best configuration in small scale
  - When exhaustive search is not possible
  - Using common application patterns
- Extend to mobile applications
  - Many modern mobile apps have patterns similar to HPC!

Cache Hierarchy Reconfiguration in Adaptive HPC Runtime Systems

Ehsan Totoni
Josep Torrellas
Laxmikant V. Kale

Charm Workshop
April 29th, 2014

PARALLEL PROGRAMMING LAB
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS
PPL UIUC

ILLINOIS