IBM T. J. Watson Research Center

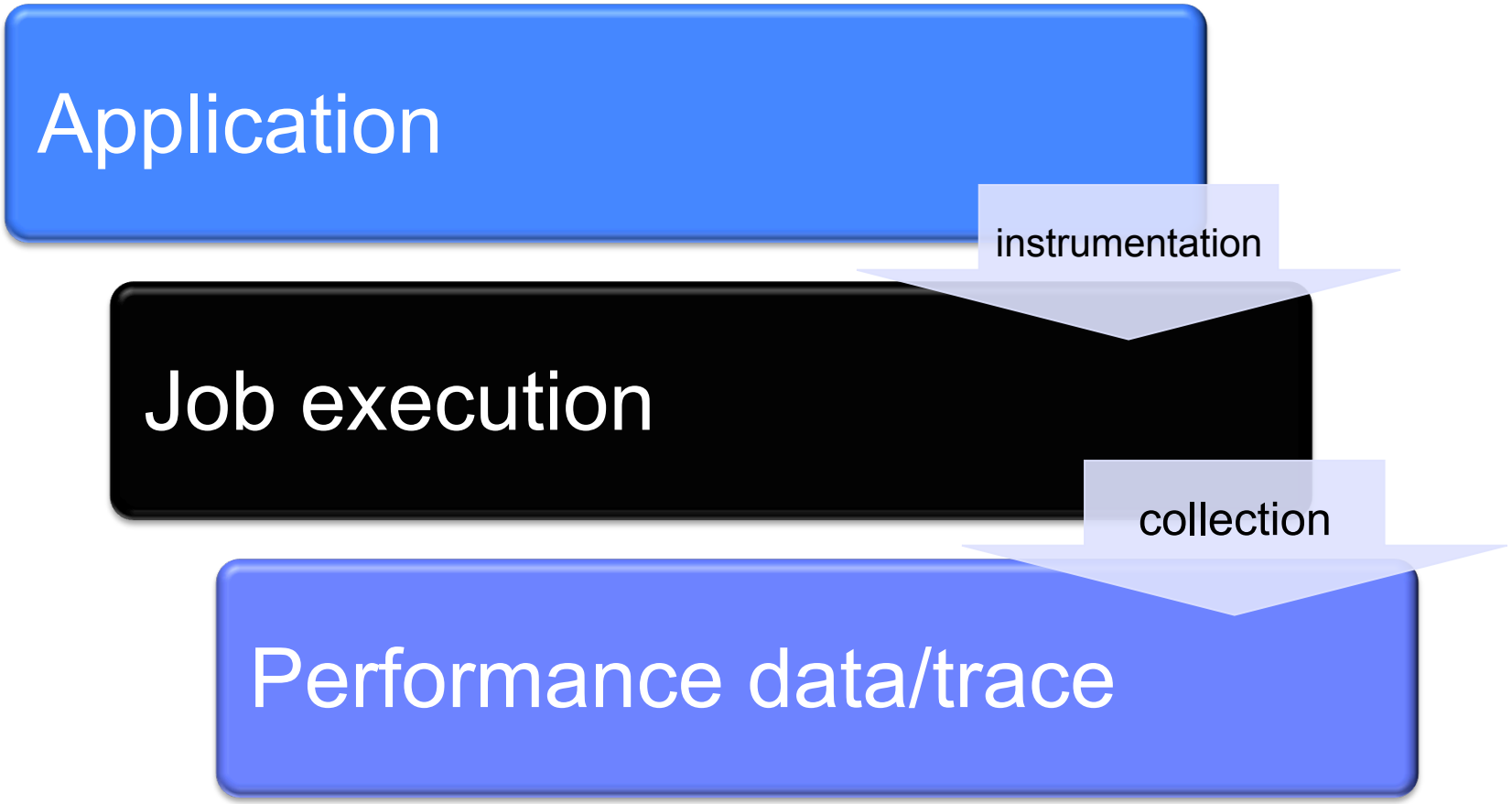
Blue Gene Performance Data Repository and Application Data Prefetching

I-Hsin Chung
ihchung@us.ibm.com

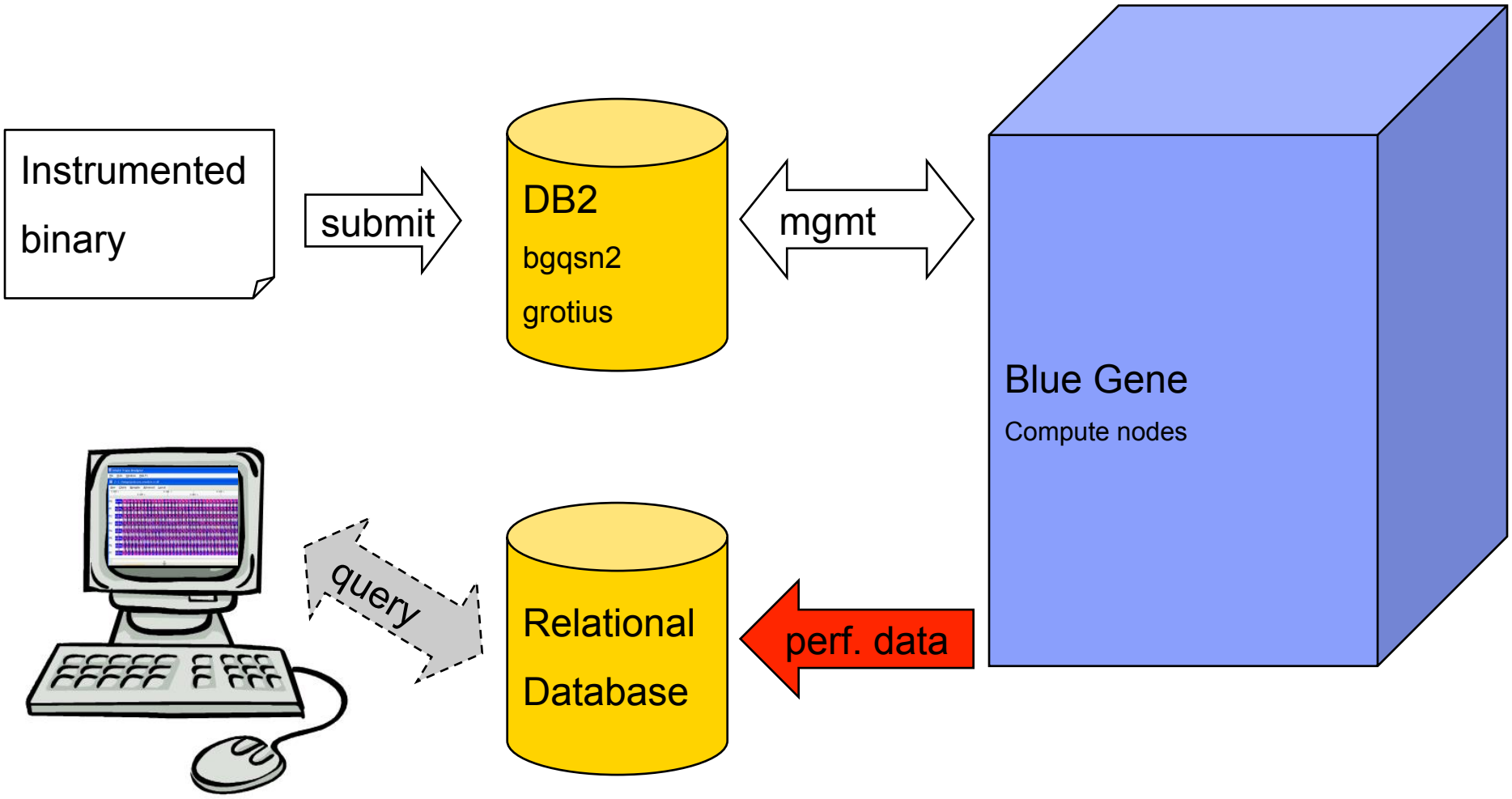
Performance Data Repository

- Goal
  - To characterize the applications on existing systems
  - To understand the system resource usage
    - To provide inputs for next generation system design
- Objective
  - Collect performance data and store them into a relational database
  - Uniform storage format
    - to support queries and presentation
    - To make comparisons cross applications or platforms

Application Performance data/trace



Performance Data Repository



How to use it?

- **Link** the application with performance tool
  - Link the profiler libraries (e.g., -lmpihpmperf or -lpomprofperf) statically
  - Or modified version of MPI compiler (wrapper)
  - Supports MPI/Hardware counter, OpenMP profiler
- **Run** the application

- **Query** the database (optional)
  - SQL statements

Link the application with performance tool – MPI/HW counter

- Example: NPB-3.3-MPI BT on grotius
- The **MPI/BGPM** library is at /gpfs/DDNgpfs1/ihchung/codes11/hpct.db2/bobmpihpm/libmpihpmperf.a

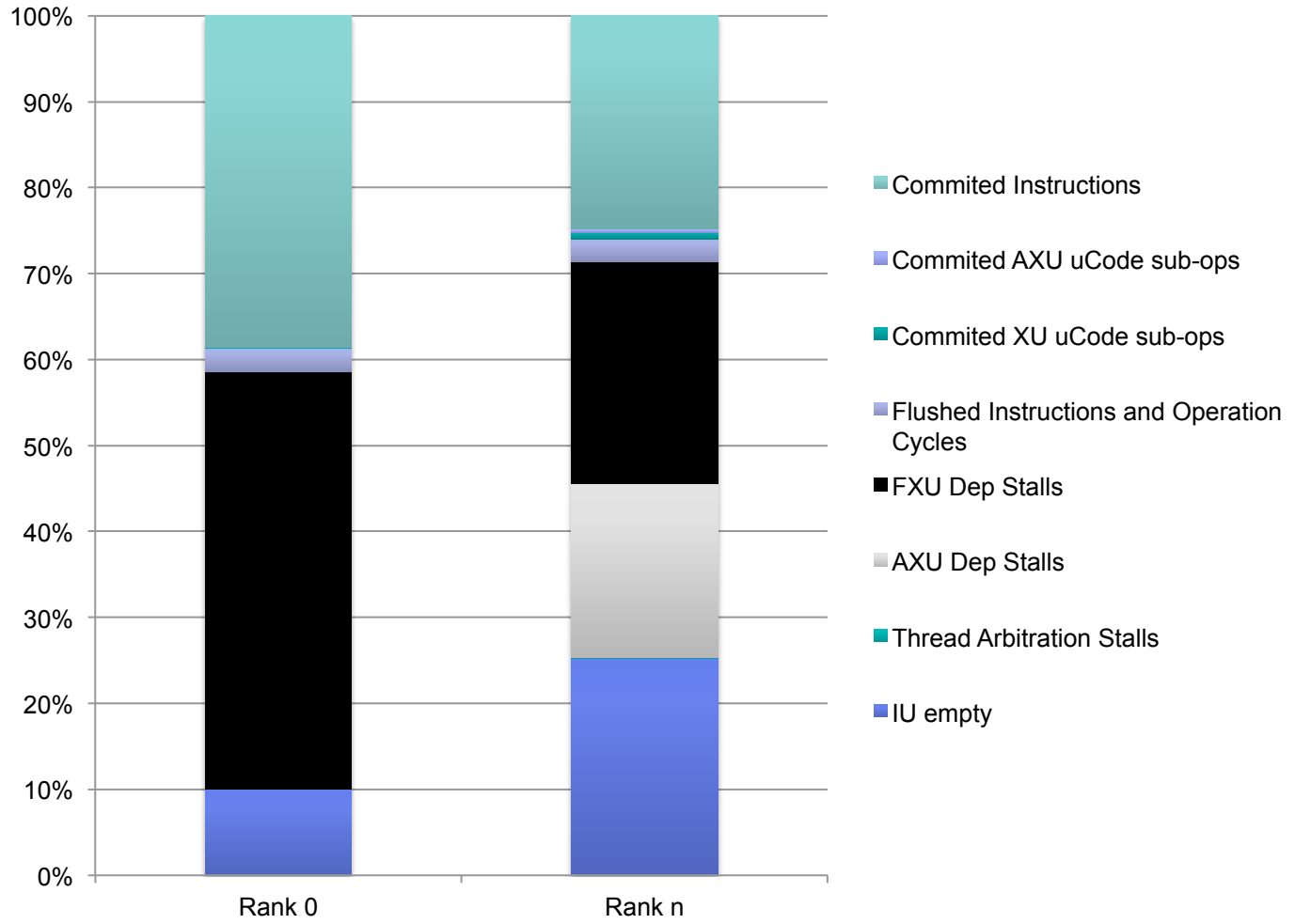
- /opt/ibmcmp/xlf/bg/14.1/bin/bgxlf_r -O -g -o ../bin/bt.A.4 bt.o make_set.o initialize.o exact_solution.o exact_rhs.o set_constants.o adi.o define.o copy_faces.o rhs.o solve_subs.o x_solve.o y_solve.o z_solve.o add.o error.o verify.o setup_mpi.o ../common/print_results.o ../common/timers.o btio.o **-L /gpfs/DDNgpfs1/ihchung/codes11/hpct.db2/bobmpihpm -lmpihpmperf** -L /bgsys/drivers/ppcfloor/comm/xl/lib -lmpich -lmpl -lopa -L /bgsys/drivers/ppcfloor/comm/sys/lib -lpami -L /bgsys/drivers/ppcfloor/spi/lib -lSPI_cnk -lrt -lstdc++

Link the application with performance tool - OpenMP

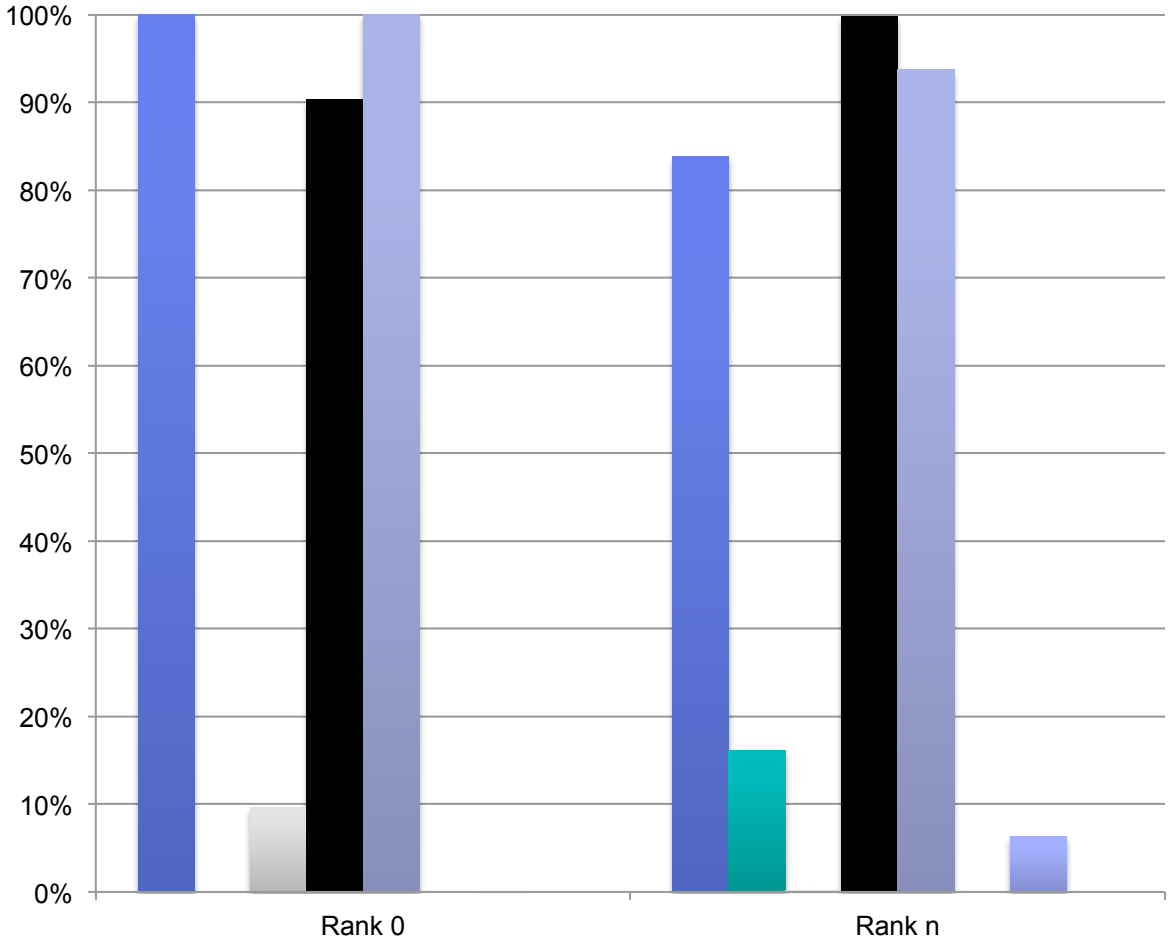
- Example: a OpenMP toy code on grotius
- The **OpenMP** profiler library is at /gpfs/DDNgpfs1/ihchung/codes11/hpct.db2/source/lib/libpomprofperf.a
- The instrumentation is done via "hooks" provided by the XL compiler

- /opt/ibmcmp/vac/bg/12.1/bin/bgxlc_r -O2 -g -qsmp=omp -qsimd=noauto -qsmp=omp myomp.c -o myomp -L/bgsys/drivers/ppcfloor/spi/lib -L . -lSPI_cnk -lxlsmp_pomp **-L../../lib -lpomprofperf**

Rosetta - CPU



Rosetta – instruction & memory



	Rank 0	Rank n
DDR bandwidth	0.001	0.005
Heap Usage	330100736	666894336
Stack Usage	20351	20351
Gflops	0	1.813
IPC	0.3869	0.2072

Rosetta – MPI comm.

Rank 0

		Call count	Data size	Time
MPI Comm	P2P	217	868	700.658
	Collective	7	84	0.034

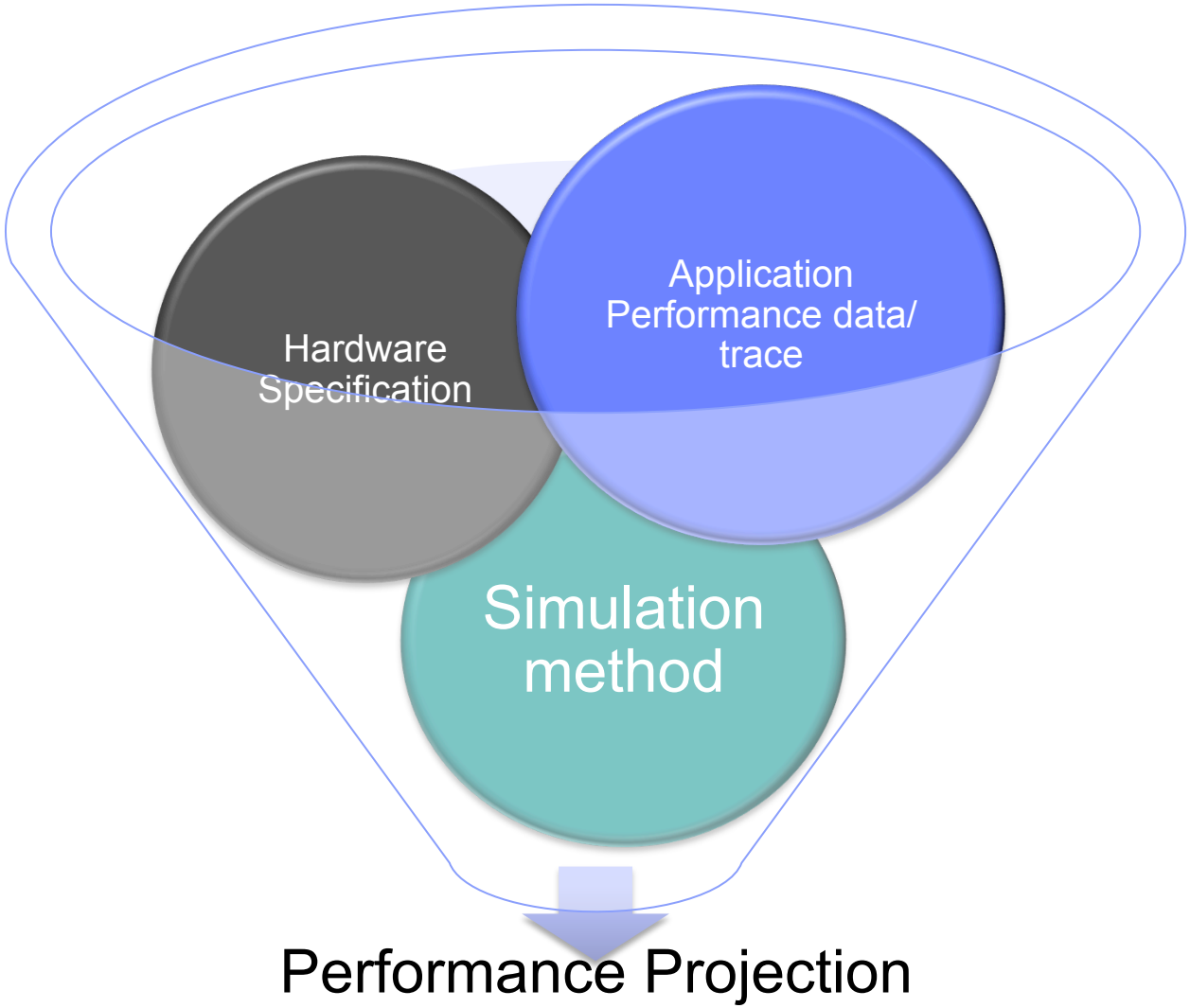
total comm	700.692
total time	711.03

Rank n

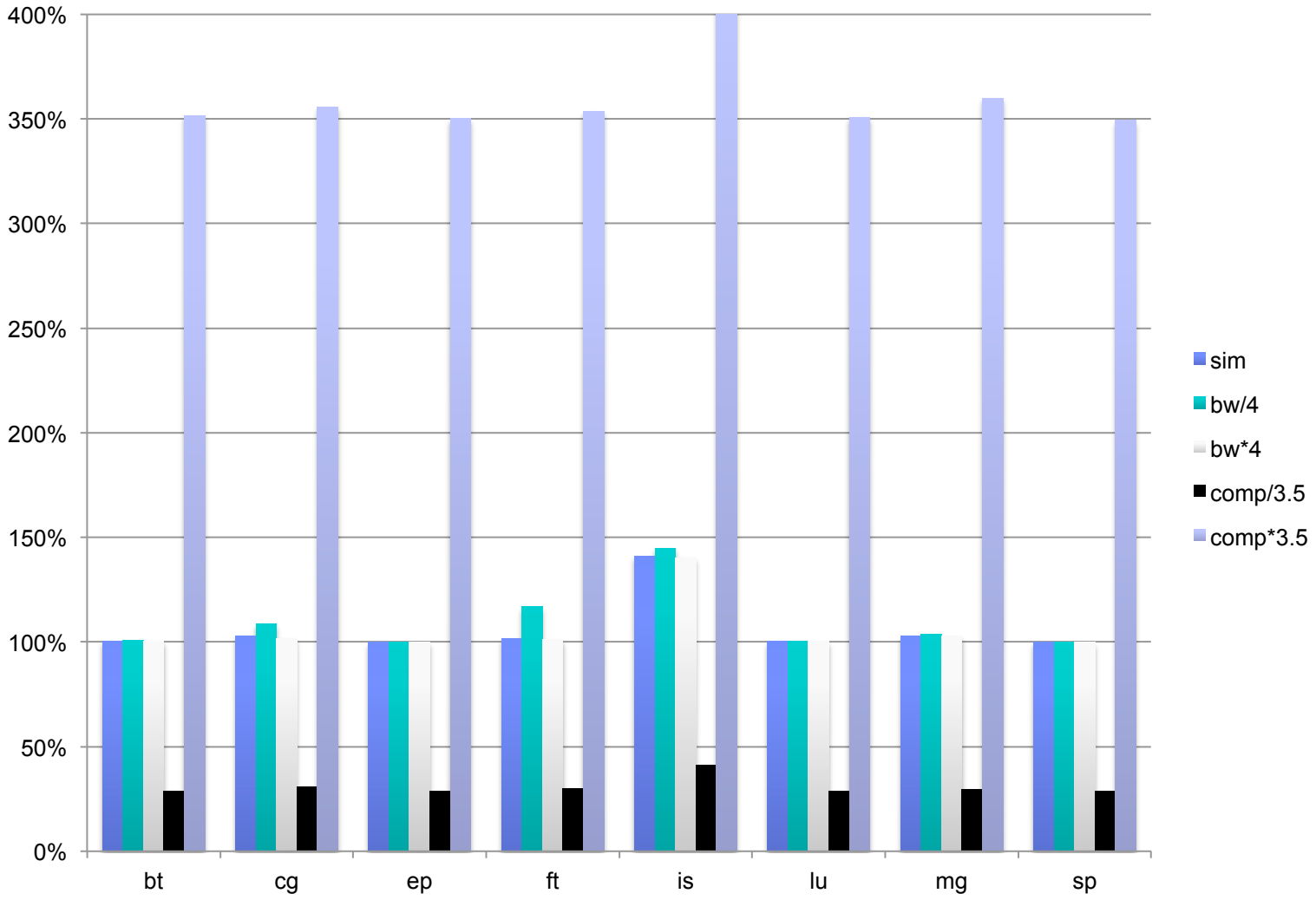
		Call count	Data size	Time
	P2P	7	28	0.022
	Collective	7	84	0.004

total comm	0.026
total time	711.03

Methodology



Simulation NPB D 64



Status

- Being deployed to ANL ALCF
- Performance data will be output into plain-text SQL files in additional to original performance data files
- User interface is being developed
  - Excel spreadsheet
  - Web interface

Application Data Prefetching

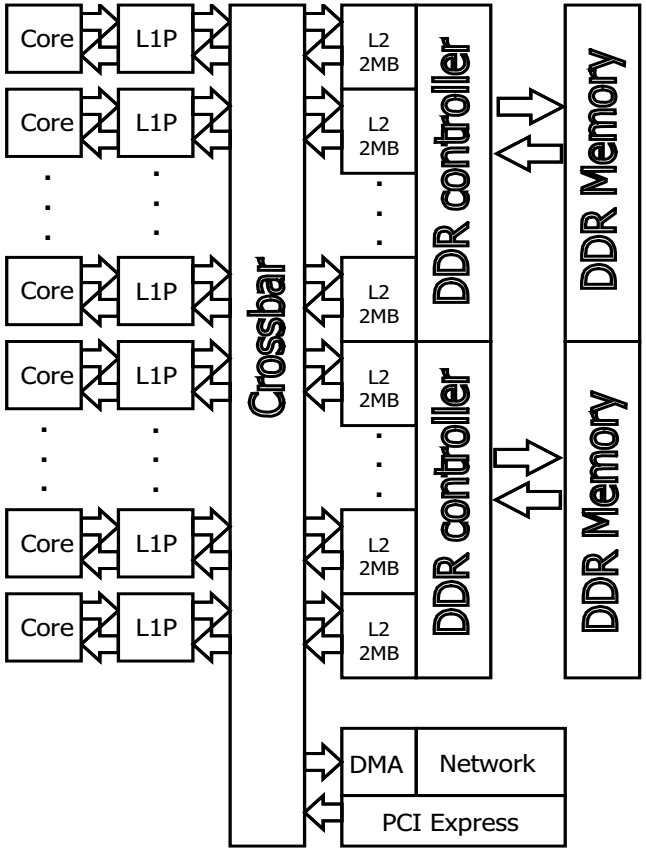
- Memory performance
  - Latency: caching, prefetching
  - Bandwidth
- Indirect memory access
  - A(B(j)): A, B arrays, j ordered index
  - Prevalent in a wide variety of science and engineering applications
  - Difficult to optimize

IBM Blue Gene/Q L1P

- Level One Cache Prefetch
  - Predicts memory access patterns
  - Prefetches the data accordingly
- Stream prefetcher
  - Handles sequential memory access patterns
- List prefetcher
  - Handles non-sequential memory access patterns

Level One Cache Prefetcher

- The interface between the core and the rest of BG/Q system
- Each core is paired with a L1P
- 4K byte L1P buffer storing data from L2
- Prefetch line size 128 bytes, 32 lines
- Buffer managed by Prefetch Directory (PFD)
- Unlike cache, L1P generates load requests autonomously

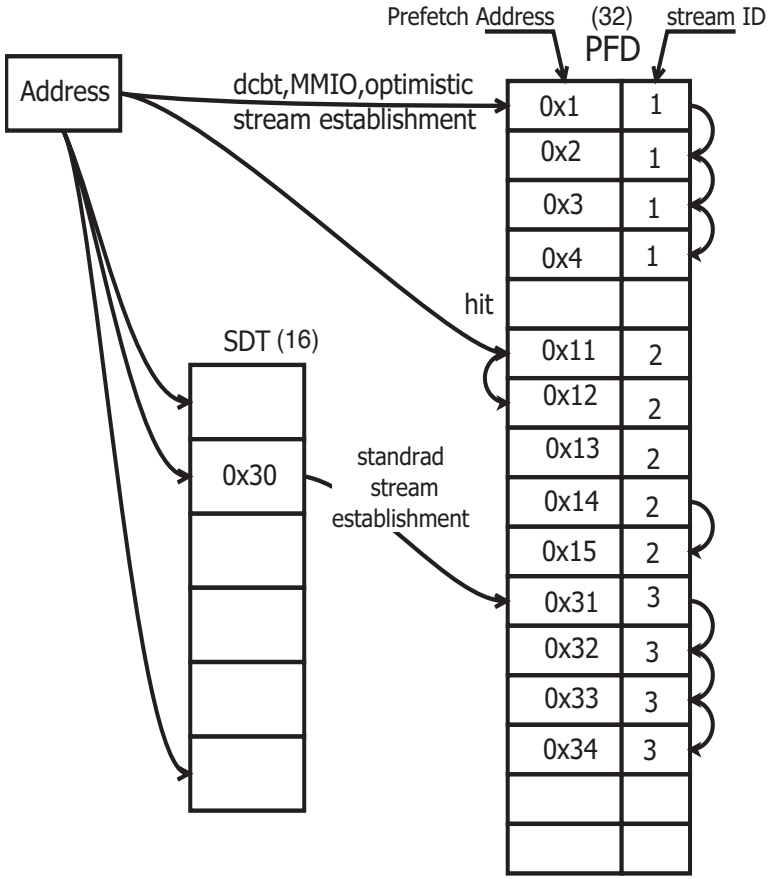


Stream Prefetch Engine

- Prefetches data from a consecutive sequence of 128-byte blocks
- Maintains up to 16 sequences (streams) per core
- Stream establishment
  - Automatic detection
    - Optimistic – no consecutive memory access pattern is required
    - Confirmed (default) – requires a detection of a stream
  - Manual setup
    - Use dcbt (data cache block touch) command
    - Write to a special memory mapped I/O (MMIO) register

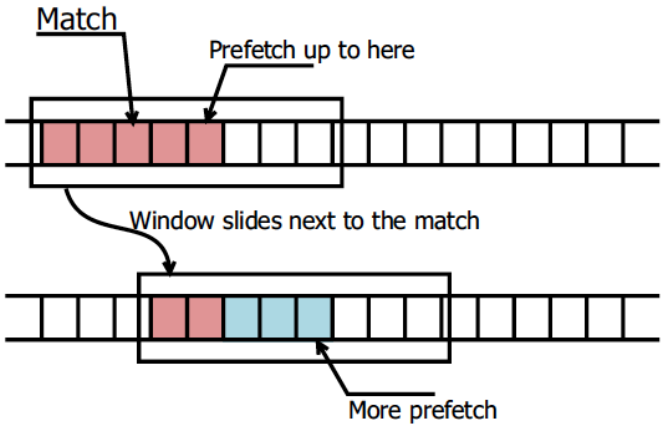
Stream Prefetch Engine - continued

- Stream detection table (SDT)
  - Holds up to 16 addresses
  - L1 miss address
    - If there is a match – stream detected
    - If no match – insert into table
- Prefetch directory (PFD)
  - Stream established – user configurable depth & stream ID
  - L1 miss address matches
    - hit: triggers further prefetching
    - not ready: increases prefetch depth



List Prefetch Engine

- Prefetches data by a predefined list of addresses
- Data prefetched up to a user-configurable depth
- Tracking location of the current L1 miss in the list
- Memory access pattern may not repeat itself exactly
  - Sliding window “ReadListArray” size of 8
- L1 miss address compared within the window
  - Found – advances the window and further prefetching
  - Not found – mismatch counter increases; too many mismatches then aborts
- Predefined list resides in main memory
  - Can be manually controlled via APIs
  - Can be used for analysis/debugging



List Prefetch Engine - continued

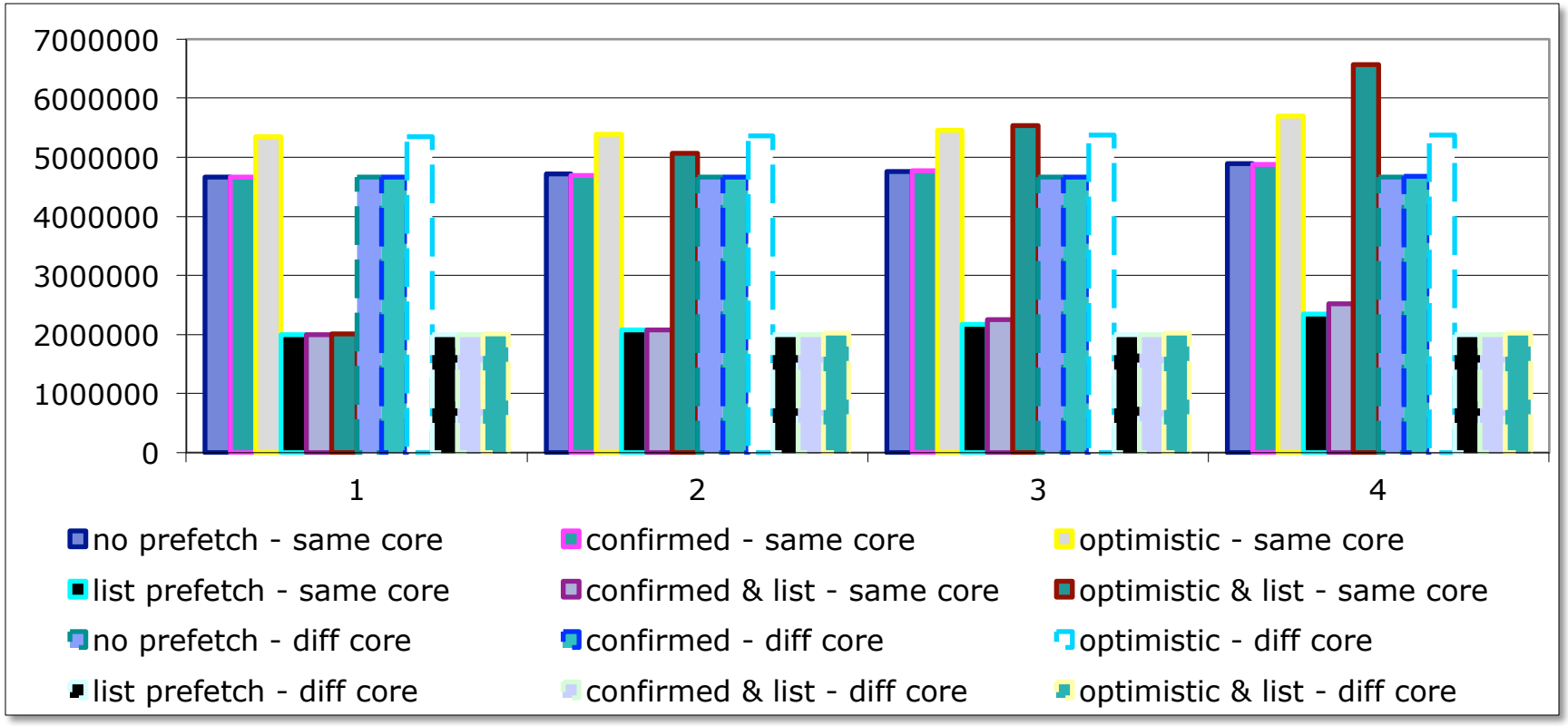
- Simple pattern compression mechanism
  - Two L1 misses fall into same L1P line
- List compression table (LCT)
- The engine can be paused/resumed
  - To exclude code segments
- Stop/deactivate list prefetch engine
  - List creating side: flush LCT, attach end-of-list mark
  - List prefetching side: wait last memory access
- One list prefetch engine per one hardware thread

Evaluation – Basic Analysis

- Three loops are used
  - Only non-uniform memory accesses
  - Half uniform and half non-uniform memory accesses
  - Only uniform memory accesses
- Up to 4 hardware threads on same/different cores
- Performance metrics
  - Time (cycles)
  - Number of L1P hits

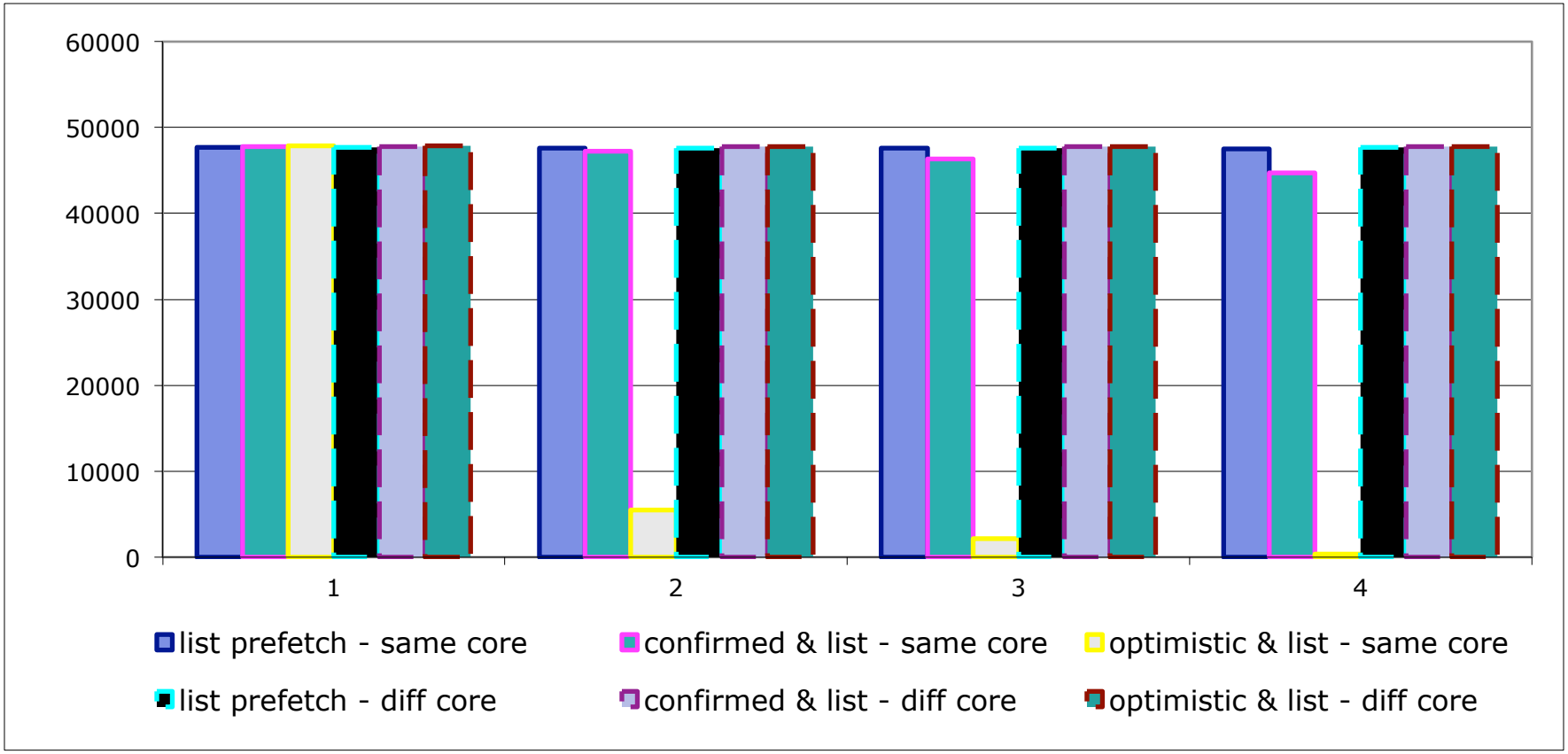
Non-uniform Memory Accesses - Time

- Weak scaling workload for threads
- Stream prefetcher is not helpful
- List prefetcher with enough resource (L1P buffer)



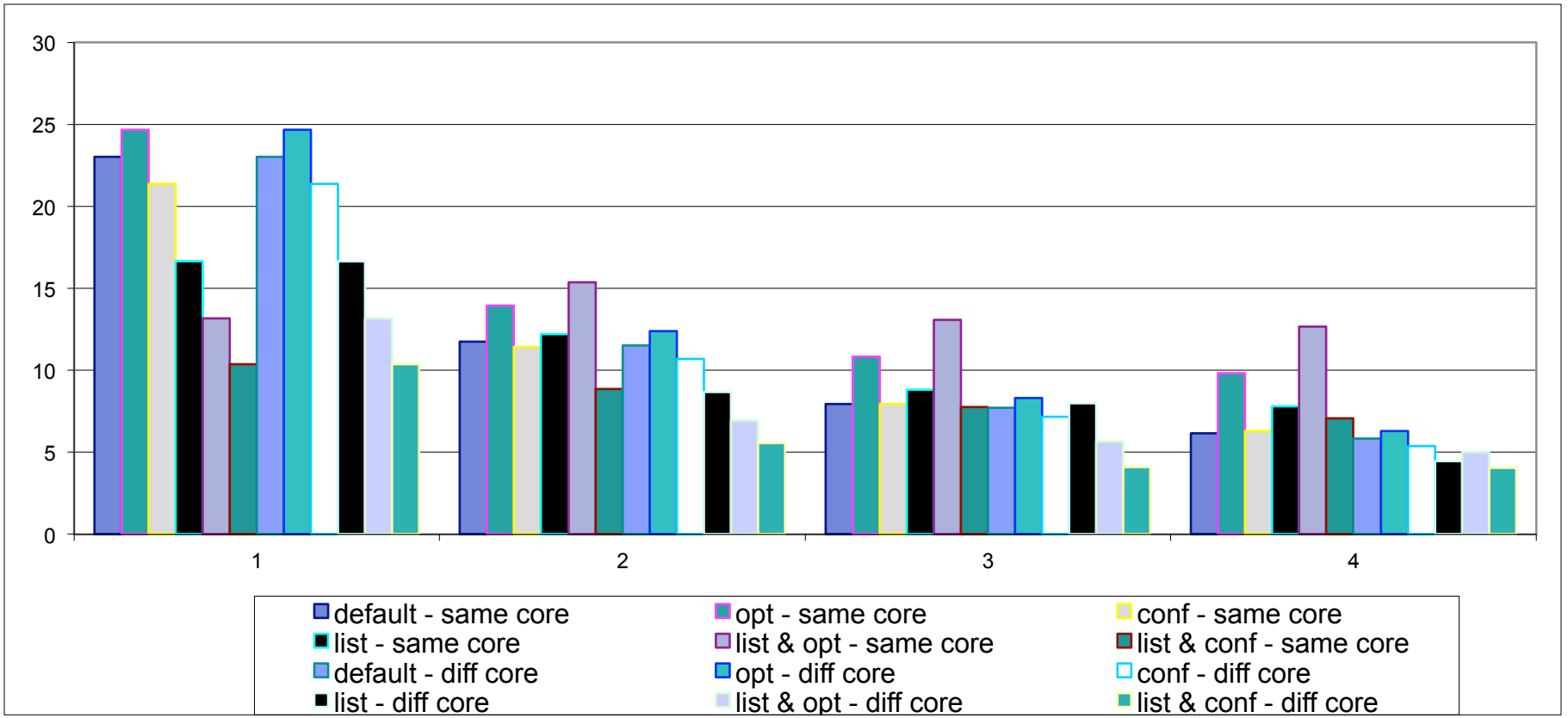
Non-uniform Memory Accesses – L1P hits

- Stream prefetcher with optimistic policy is too aggressive when competing L1P buffer with list prefetcher



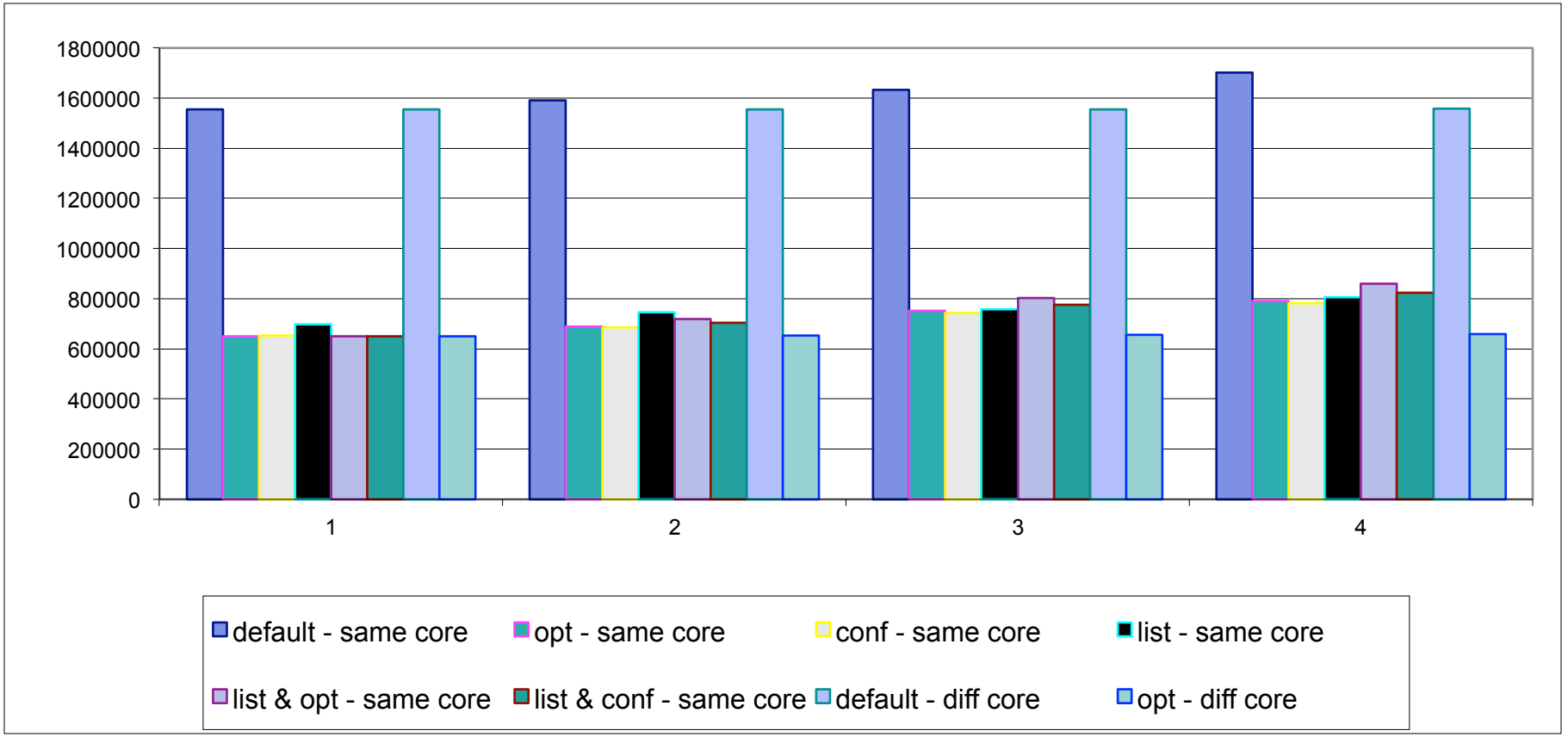
Half Uniform and Half Non-uniform Memory Accesses - Time

- Strong scaling workload for threads
- Performance further improved when both prefetchers working together



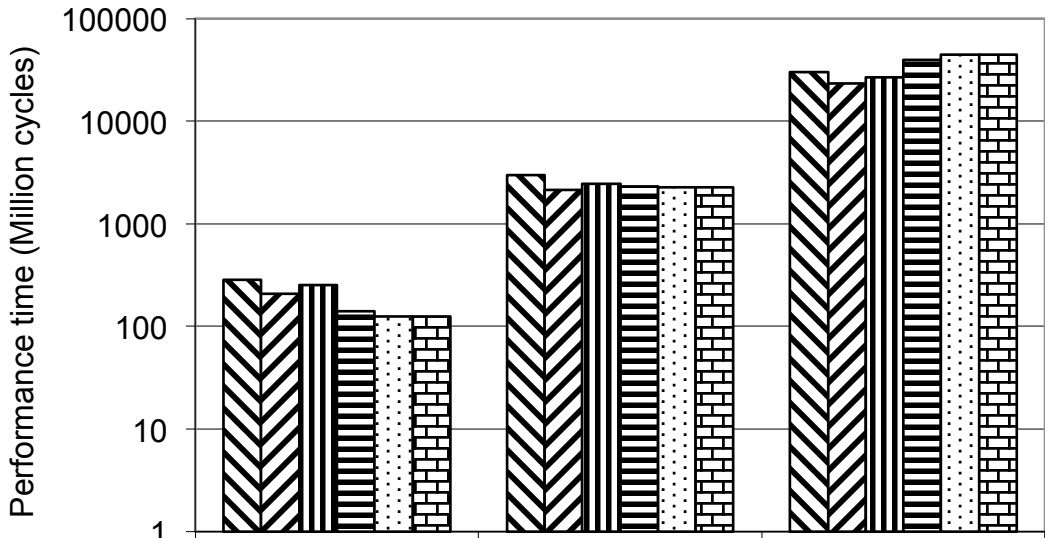
Uniform Memory Accesses - Time

- Weak scaling workload for threads
- List prefetcher is competitive to the stream prefetcher to achieve similar performance



Evaluation – Graph Algorithm

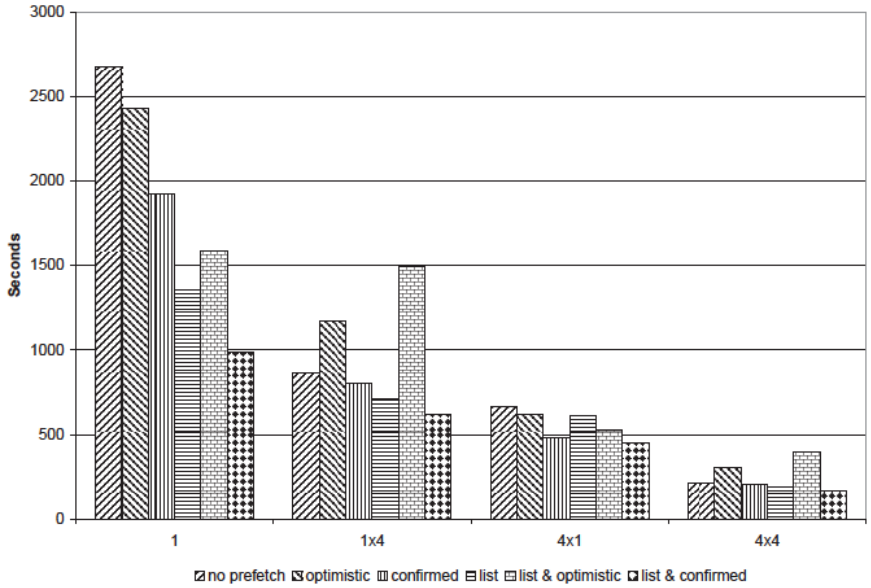
- An Variant of Shiloach-Vishkin Algorithm (SV)
- A representation of connectivity algorithms adapts the widely-used graft-and-shortcut approach
- Complexities of O[log n] time and O[(m+n) log n] work under CRCW PRAM model
- Grafting components dominates execution time
- Indirect memory accesses
- Works reasonably well with small to mid size datasets
- One may break a big graph into small pieces and handles separately



	100K, 400K	1M, 4M	10M, 40M
sw + opt	285	2960	30186
sw + conf	210	2138	23201
sw + dcbt	253	2460	26749
list + opt	141	2299	39611
list + conf	124	2280	44130
list + dcbt	124	2280	44130

Evaluation – CG Iteration and Sparse Matrix

- The conjugate gradient method is frequently used to solve linear systems
- The CG method requires iteration of sparse matrix and vector for multiplication
- Sparse matrix is represented in CSR format (compressed sparse row);
- NPB CG OpenMP 3.3 class C
- CG Kernel 60% strided memory accesses, 40% random
  - Uniform: accessing matrix elements (8-byte) and column indices (4-byte) in CSR
  - Random: accessing vector elements (8-byte) by the column indices
  - Strong scaling
- Optimistic stream policy is too aggressive when with limited resource



Discussion

- With sufficient of hardware resource and time, L1P prefetches data from L2 into L1P and improves the performance
  - Overlapping computation with data prefetching
- If data requests from L1 misses are predicted correctly
  - It reduces 2/3 of the latency
  - 18 more cycles (between L1 and L1P) to be overlapped
- The stream of prefetch list addresses go through L2 to main memory
  - The list prefetch reduces L1 cache miss penalty at the cost of the memory bandwidth (to L2 and/ or main memory) and the storage space
  - Unless heavily memory-bound application, list prefetching should incur little overhead

L1	6 cycles
L1P	24 cycles
L2	84 cycles
Memory	346 cycles

Discussion - continued

- L1P buffer (4K bytes) is shared between stream and list prefetch engines up to four hardware threads
  - Competition for injecting a request into PFD (prefetch directory)
  - Stream prefetch has priority when requests come at the same time
  - Competition may cause thrashing
- Size of the sliding window
  - Decides mismatch of addresses allowed
  - Constrained by the hardware (parallel comparison)
  - May be eased by using multiple smaller lists
- List of addresses provides a channel for performance tuning
  - Analysis together with hardware performance counter and debugging registers

Conclusion

- Performance Data Repository aims to
  - Characterize the applications
  - Study the hardware usage
- L1P is intended to benefit applications with performance limited by indirect or random memory access patterns
- L1P shows effectiveness in graph-based applications and sparse matrix solvers with sufficient hardware resource.
- List prefetcher works well with stream prefetcher
- Future work
  - More intelligent control
  - Coordination with other features such as massive speculative threading