【ICDE2013 勉強会】

R10: Main Memory Query Processing

担当：山室健

概要

- このセクションの特徴
  - in-memoryを前提としたクエリ最適化(Hash Joinの高速化やMVによる資源の利活用)に関する話題

- 紹介する論文リスト
  - 1. Efficient Many-Core Query Execution in Main Memory Column-Stores
  - 2. Recycling in Pipelined Query Evaluation
  - 3. Main-Memory Hash Joins on Multi-Core CPUs: Tuning to the Underling Hardware

Disk-less Modern Database Management

Publickey Enterprise IT × Cloud Computing × Web Technology / Blog

Oracle 12.1cはカラム型インメモリデータベースになると
エリソン氏。32TBメモリのSPARCサーバと組み合わせ

2013年6月25日

http://....から引用

1. Efficient Many-Core Query Execution in Main Memory Column-Stores

Jonathan Dees (SAP) Peter Sanders (KIT - Karlsruher Institut fuer Technology)

1. Efficient Many-Core Query Execution in Main Memory Column-Stores

- A overview
  - 現状提案されている様々なModernなDB最適化手法を用いてOLAP向けのクエリ(TPC-H)の高速化をSAP HANA上で実現しました、という実装論文

- Contribution
  - ModernなDB最適化手法が広く浅く議論されている
  - OLAP向けに最適化された既存のColumnDB(Vectorwise)と比較して10～200倍の高速化(at SF100 and SF300)

1. Efficient Many-Core Query Execution in Main Memory Column-Stores

- 扱っているDB最適化手法
  - Just-in-time SQL Compilation
  - Vector-at-a-time Query Execution
  - Column-oriented Storage
  - NUMA-based Thread Schedulers/Indices
  - Inverted Indices
  - Block Summaries like Zonemaps in Netezza
  - Loop Unrolling/SIMD Instructions
  - ....

1. Efficient Many-Core Query Execution in Main Memory Column-Stores

- Experimental
  - 22 Queries in TPC-H
  - 4 Xeon X7560
    - 8-cores at 2.27 GHz.
  - 256GiB Mem
    - Mem bandwidth 50GiB/s
- Memory Space Used
  - データ圧縮率51%
  - 索引による余剰領域19%

TABLE II
POWERTEST RUNNING TIMES: WE VS. TPC-H RECORD HOLDER.
F = SPEEDUP OVER OTHER SYSTEM. S = SCALED BY SPECINTRATE.

Query	SF100				SF300			
	We [s]	Vector Wise [s]	F	S	We [s]	Vector Wise [s]	F	S
1	0.20	1.2	6	6	0.60	3.6	6	6
2	0.01	0.3	30	28	0.01	0.7	100	105
3	0.10	0.2	2	2	0.25	0.5	3	3
4	0.01	0.1	10	9	0.01	0.3	30	32
5	0.05	0.7	14	13	0.41	2.1	4	4
6	0.03	0.1	3	3	0.11	0.2	3	3
7	0.10	0.5	5	5	0.39	1.4	4	4
8	0.02	0.8	40	38	0.02	1.9	125	132
9	0.18	2.7	15	14	0.50	9.5	24	26
10	0.06	1.9	32	30	0.15	5.1	51	53
11	0.03	0.3	10	9	0.08	0.9	19	20
12	0.05	0.3	6	6	0.30	0.7	4	4
13	0.08	3.7	46	44	0.17	11.3	69	73
14	0.02	0.6	30	28	0.24	1.7	10	10
15	0.07	0.3	4	4	0.38	0.7	3	3
16	0.19	0.9	5	4	0.55	2.2	6	6
17	0.01	0.5	50	47	0.01	1.5	190	200
18	0.08	1.6	20	19	0.21	5.1	29	31
19	0.01	1.5	150	142	0.03	3.9	93	98
20	0.02	0.8	40	38	0.05	2.1	50	53
21	0.12	2.9	24	23	0.54	8.2	17	18
22	0.02	0.7	35	33	0.07	2.2	31	33
$\sum$ [s]	1.4	22.6			5.08	65.8		
avg.	0.07	1.03	26	25	0.23	3.50	34	32
SPEC Rate	710	670			710	670		
cores	32	12			32	12		
RAM [GB]	256	192			256	384		

※論文内TableⅡから引用

1. Efficient Many-Core Query Execution in Main Memory Column-Stores

処理を並列化しない場合の性能比較 TABLE III

POWERTEST ON REGULAR DESKTOP MACHINES. WE IN MS, SINGLE THREAD AND ALL OTHER RESULTS IN MULTIPLE OF OUR RUNNING TIME.

Query	SF1									SF10	
	We	speedup over 1 core	Vector Wise 2.5	Monet DB 5	MonetDB /X100 [3]	Postgre SQL untuned	My SQL untuned	HI-QUE [12]	Hy-Per [13]	We	speedup over 1 core
	[ms]	in multiples of We								[ms]	in mult.
1	13.8	3.4	10	28	23	2 770	1 480	26	2.5*	136.2	3.5
2	0.3	1.0	237	290	33	1 670	2 330		417*	0.7	2.0
3	0.4	2.8	228	198	50	29 500	19 800	1 030	200*	4.1	3.2
4	0.2	3.0	395	890	100	3 000	6 500		585*	1.9	3.7
5	1.4	3.4	69	68	29	286	9 210		789*	21.3	3.5
6	2.1	3.4	12	53	10	1 290	1 710			20.7	3.4
7	3.5	3.1	35	30	63	1 540	1 090			43.5	3.4
8	0.3	3.0	407	403	100	10 700	2 670			5.2	3.3
9	4.6	3.3	34	28	96	4 590	1 130			79.9	3.5
10	1.7	1.8	65	44	112	294	4 530	571		12.2	2.5
11	0.6	2.3	252	47	33	983	433			8.2	1.9
12	2.6	3.4	46	93	15	1 420	1 310			25.5	3.4
13	2.6	3.5	47	1423	400	1 810	2 150			28.8	3.6
14	0.4	3.0	98	150	50	6 750	45 000			4.3	3.5
15	1.9	2.7	44	39	21	1 370	3 630			17.0	3.2
16	7.9	2.1	11	21	18	646	671			33.4	3.2
17	0.1	1.0	730	1500	200	98 M	6 000			0.3	3.0
18	5.9	3.5	40	92	19	4 580				57.6	3.5
19	0.3	3.0	230	1320	167	14 700	600			3.4	3.6
20	0.5	3.2	164	136	100	31 M	400			5.0	3.3
21	4.0	2.9	65	118	43	6 000	875			40.5	3.0
22	0.5	2.8	140	236	80	3.5 M	540			5.1	2.9
∑	56 ms	170 ms	2.4 s	7.3 s	3.0 s	≈ 8 h	108.5 s			0.56 s	1.87 s
avg.	2.5 ms	7.7 ms	109 ms	336 ms	138 ms					25.2 ms	85 ms
avg. × faster	1	2.8	152	328	80					1	3.2
SPEC											
Rate	92.4		92.4	92.4	≈ 10	54	54	21	133	92.4	
Int	30.5	30.5	30.5	30.5	≈ 10	16.5	16.5	12.3	32.0	30.5	30.5
cores	4	4	4	4	1	4	4	2	4	4	4

※論文内TableⅢから引用

2. Recycling in Pipelined Query Evaluation

Fabian Nagel (The University of Edinburgh) Peter Boncz (CWI) Stratis Viglas (The University of Edinburgh)

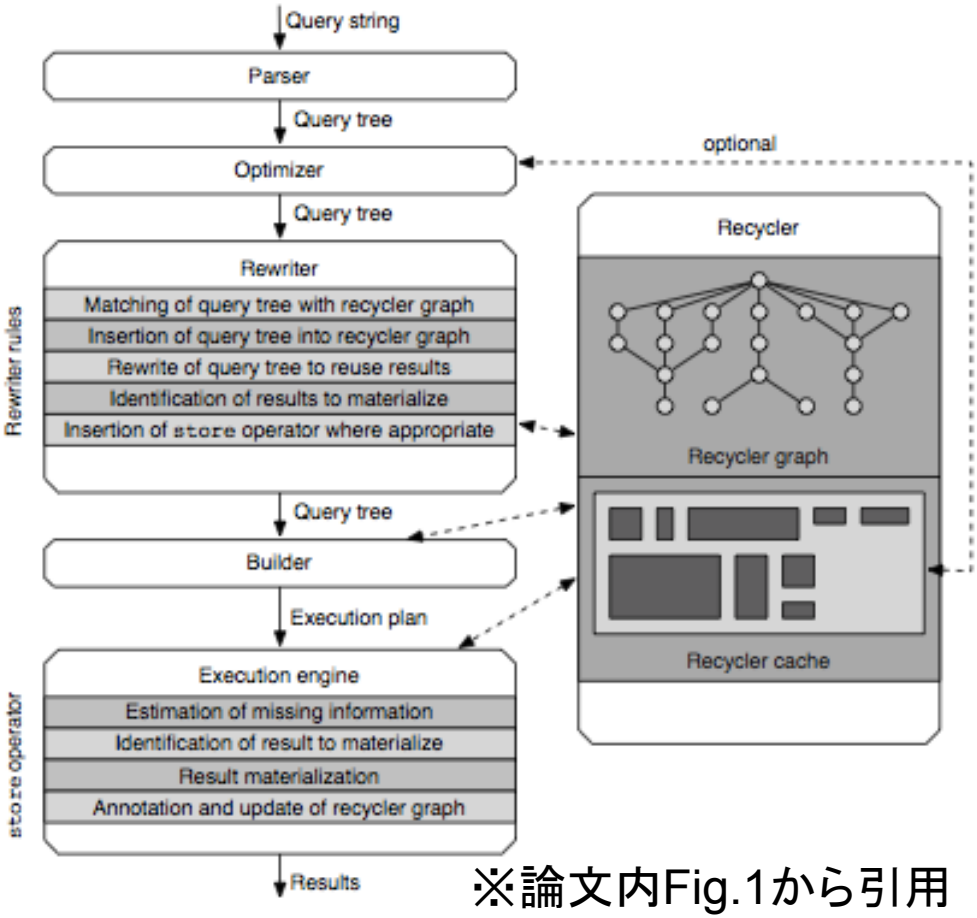
2. Recycling in Pipelined Query Evaluation

- A overview
  - クエリ実行中に発生した中間結果、もしくは最終結果（Materialized View）をクエリ間で自律的に再利用（Recycle）することで、全体の処理性能を改善

- Contribution
  - パイプライン実行モデル（Volcano-style）と、クエリの中間/最終結果を再利用するスキームを統一的に扱う手法の提案
    - Recycler Graph（DAG）とRecycler CacheによるMVの管理
    - 利得（Benefit）関数によるクエリのRewirte処理
  - 実際にVectorwise[2][20]上に実装し評価

2. Recycling in Pipelined Query Evaluation

- MVのRecycling
  - 最適化後のクエリ木とRecycler Graphを比較してRewriterで書き換え
    - Recycle Graphを考慮したOptimizerでの最適化は行わない（optional）
  - Execution Engine上で発生したMVを用いてRecyclerを更新
    - 各MVの参照統計情報
    - Recycler Graphの更新とCacheの入れ替え

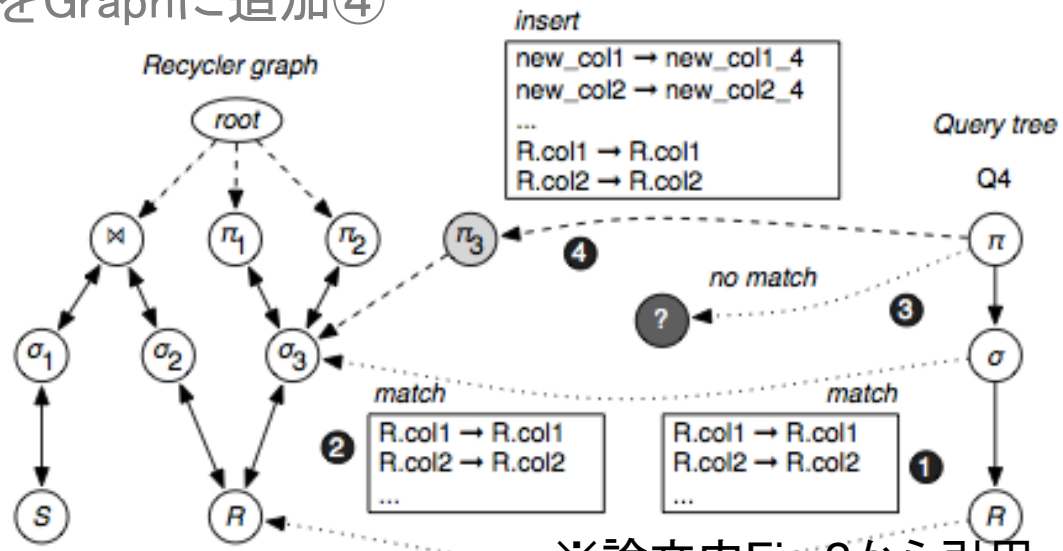


※論文内Fig.1から引用

Fig. 1. The high-level architecture of the recycler

2. Recycling in Pipelined Query Evaluation

- MVのMatchingとInsertion
  - クエリ木とRecycler GraphをBottom-up的にMatching
    - Graphの各ノードはOperator（LeafはRelation）で、ノード参照はHashを用いることでO(1)の探索①②③
    - Graph上に存在しない候補が存在した場合には、クエリ実行中に新規エントリをGraphに追加④



※論文内Fig.2から引用

Fig. 2. An example of the matching procedure

2. Recycling in Pipelined Query Evaluation

- TPC-H Experiments
  - 3つのRecycle Strategyで評価
    - HIST: 中間/最終結果をCacheに入れるかどうかの判断は過去のクエリ履歴のみから判断(2回以上使用しないとRecycleされない)
    - SPEC: 初回出現時に投機的にCacheに追加
    - PA: Top-kクエリやselectionを投機的にCacheに追加

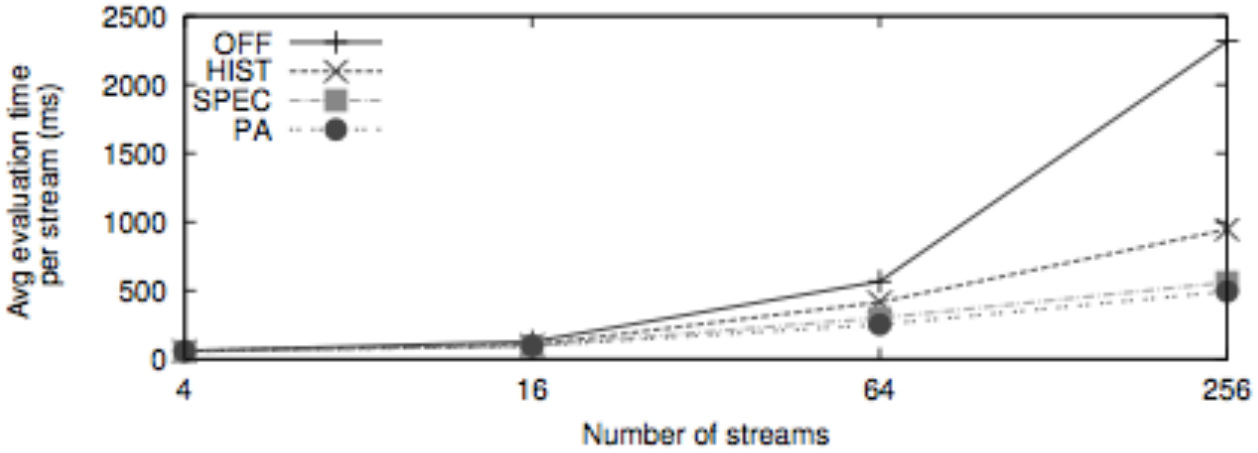


Fig. 7. Average time per TPC-H stream

2. Recycling in Pipelined Query Evaluation

- TPC-H Experiments
  - 3つのRecycle Strategyで評価

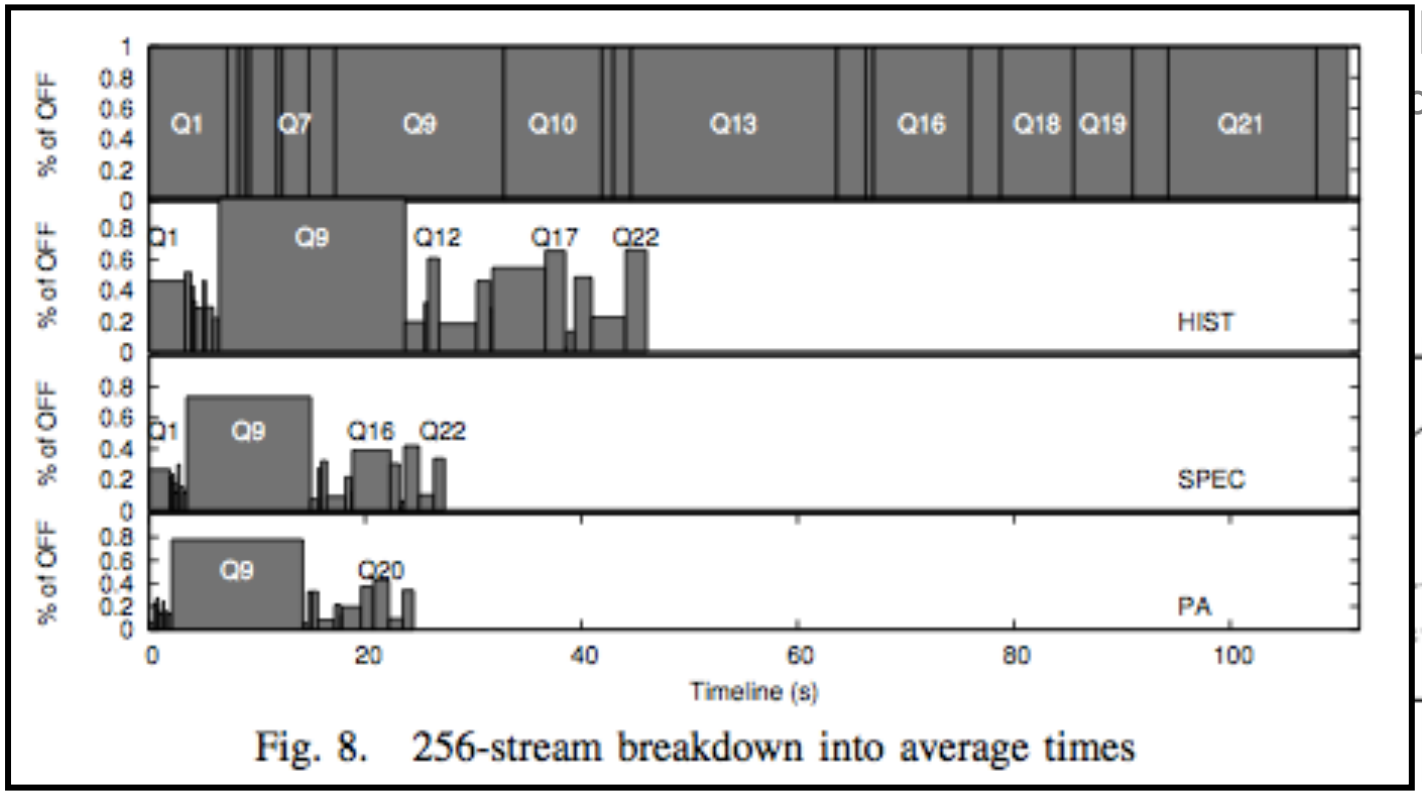


Fig. 8. 256-stream breakdown into average times

断は過去の
cleされない）

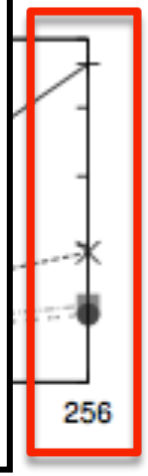


Fig. 7. Average time per TPC-H stream

3. Main-Memory Hash Joins on Multi-Core CPUs: Tuning to the Underling Hardware

Cagri Balkesen (ETH Zurich) Jens Teubner (ETH Zurich) Gustavo Alonso (ETH Zurich) M. Tamer Özsu (University of Waterloo)

3. Main-memory Hash Joins on Multi-Cores CPUs

- A overview
  - cache-oblivious/consciousなHashJoins手法の性能比較
    - cache-oblivious*: HW最適化のためのknobが無い手法
    - cache-conscious: ある特定のHW環境に最適化された手法

*http://en.wikipedia.org/wiki/Cache-oblivious_algorithm

- Contribution
  - 既存手法を分析し、HW環境に適した最適化を提案
  - 上記2つの手法（oblivious/conscious）とHW環境の依存関係を分析/再考し、HW最適化の基礎を構築
  - HW最適化の重要性を示唆（考慮の有無で3～6xの性能差）
  - SC: http://www.systems.ethz.ch/projects/paralleljoins

3. Main-memory Hash Joins on Multi-Cores CPUs

- 取り扱う2種類のHashJoins*

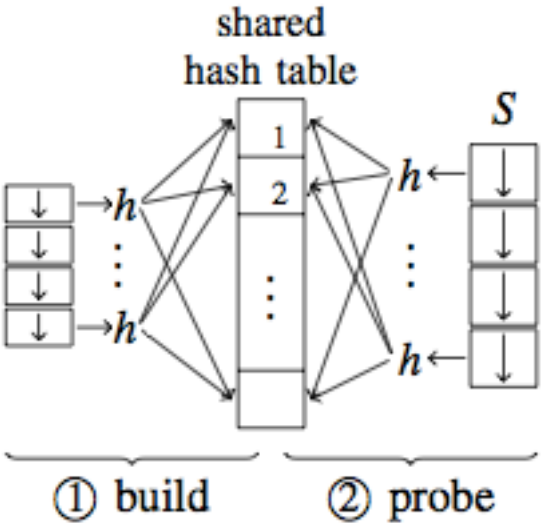


Fig. 2. *o partitionin* join.

No-Partitioning Joins
(Cache-oblivious way)
※論文内Fig.2から引用

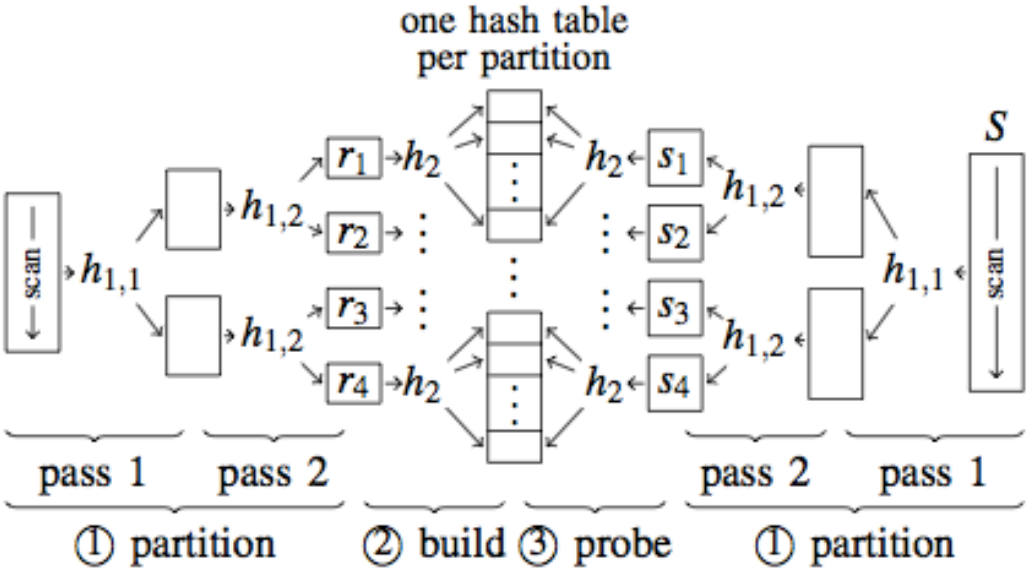


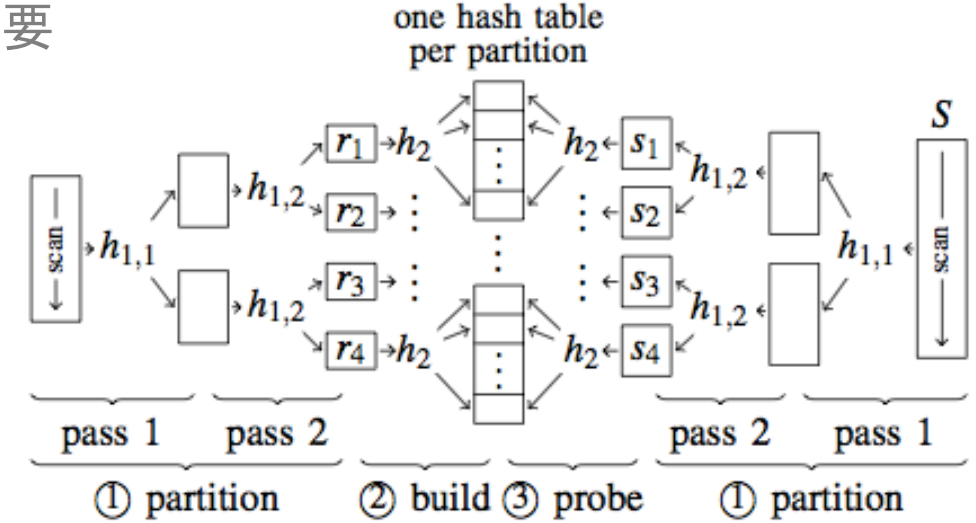
Fig. 4. *adi oin* (as proposed by Manegold et al. [4]).

Radix Joins
(Cache-conscious way)
※論文内Fig.4から引用

*1-pass Algorithmを前提

3. Main-memory Hash Joins on Multi-Cores CPUs

- Radix Joins（Cache-conscious）
  - CPUキャッシュ効率を考慮してHashTableを複数分割
    - 各passで$2^{\#\text{radix bits}}$に分割
    - キャッシュの観点から極力小さいほうが良い
  - ただし、小さすぎるとTLB*によるペナルティが顕在化[4]
    - HWを考慮した最適化が必要



*Translation Look-aside Buffer, 物理ページaddr.のvirtual/physical変換表

3. Main-memory Hash Joins on Multi-Cores CPUs

- Experimental
  - サイズの偏りを考慮した2パタンのデータA/Bを用意
  - 異なるCPUを4パタン用意

TABLE I
WORKLOAD CHARACTERISTICS

	A (from [2])	**B** (from [1])
size of *ey*/*payload*	8/8 bytes	4/4 bytes
size of	$16 \cdot 2^{20}$ tuples	$128 \cdot 10^6$ tuples
size of S	$256 \cdot 2^{20}$ tuples	$128 \cdot 10^6$ tuples
total size	256 MiB	977 MiB
total size S	4096 MiB	977 MiB

※論文内Table Ⅰから引用

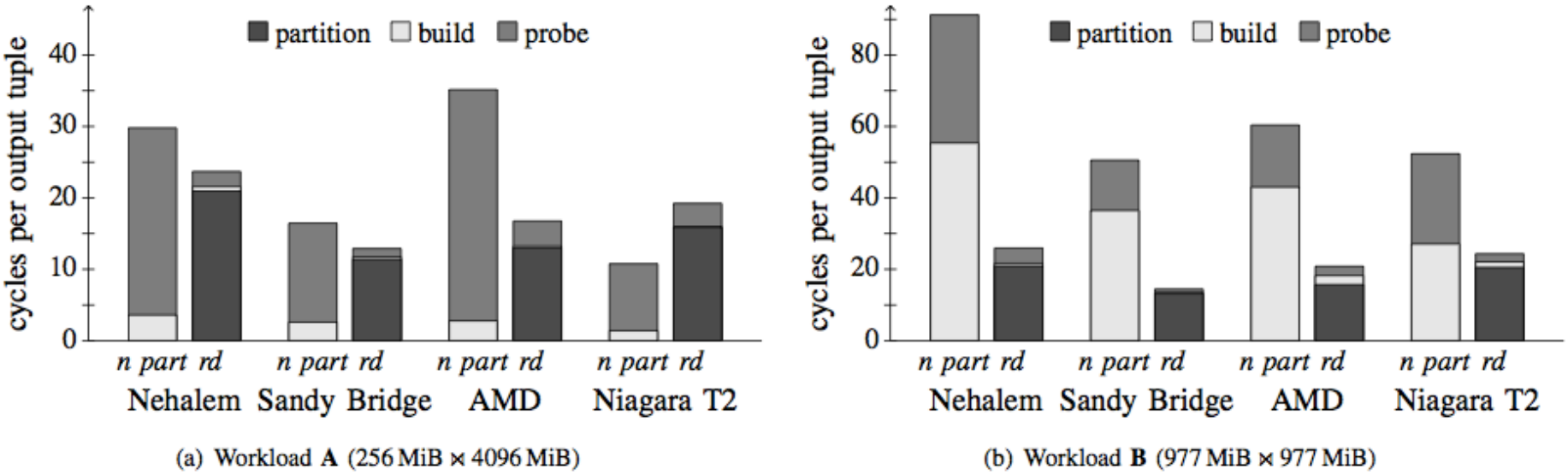
TABLE II
HARDWARE PLATFORMS USED IN OUR EVALUATION

	Intel Nehalem	**Intel Sandy Bridge**	**AMD Bulldozer**	**Sun Niagara 2**
CPU	Xeon L5520 2.26 GHz	Xeon E5-2680 2.7 GHz	Opteron 6276 2.3 GHz	UltraSPARC T2 1.2 GHz
Cores/Threads	4/8	8/16	16/16	8/64
Cache sizes (L1/L2/L3)	32 KiB 256 KiB 8 MiB	32 KiB 256 KiB 20 MiB	16 KiB 2 MiB 16 MiB	8 KiB 4 MiB -
TLB (L1/L2)	64/512	64/512	32/1024	128/-
Memory	24 GiB DDR3 1066 MHz	32 GiB DDR3 1600 MHz	32 GiB DDR3 1333 MHz	16 GiB FBDIMM
VM Page size	4 KiB	4 KiB	4 KiB	8 KiB

※論文内Table Ⅱから引用

3. Main-memory Hash Joins on Multi-Cores CPUs

- Experimental
  - データA/BにおけるCPU時間評価
    - No-Partitioning Joinの処理比率(B/P)は入力サイズに依存
    - Radix Joinの大半の時間はPartition処理



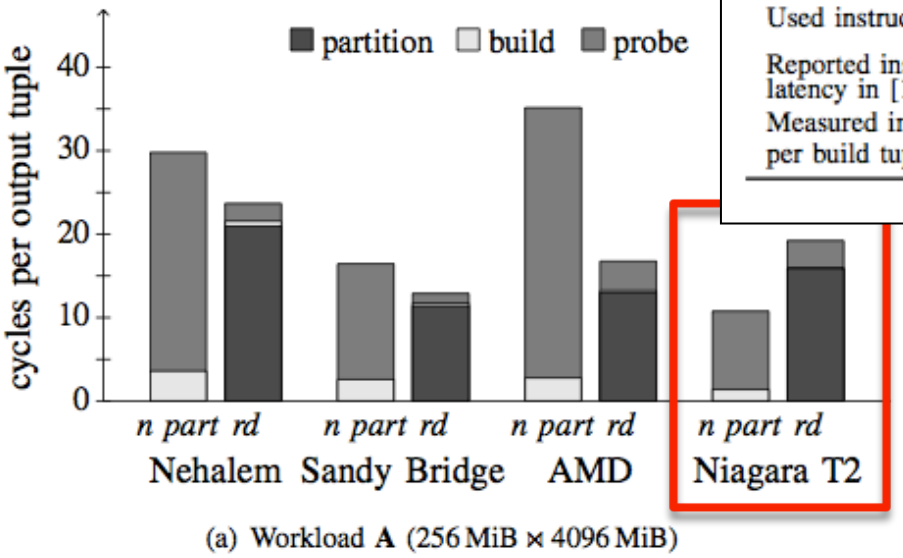
※論文内Fig.14から引用

3. Main-memory Hash Joins on Multi-Cores CPUs

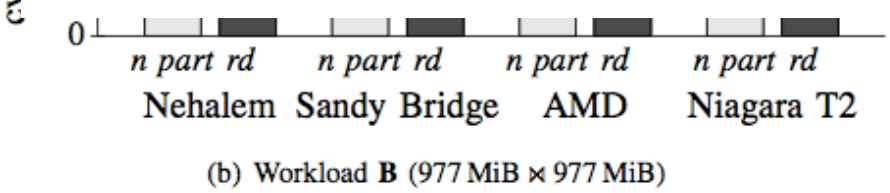
- Experimental
  - データA/BにおけるCPU時間評価
    - No-Partitioning Joinの
    - Radix Joinの大半の時

TABLE V
LATCH COST PER BUILD TUPLE IN DIFFERENT MACHINES

	Nehalem	Sandy Bridge	Bulldozer	Niagara 2
Used instruction	xchgb	xchgb	xchgb	ldstub
Reported instruction latency in [14], [15]	~20 cycles	~25 cycles	~50 cycles	3 cycles
Measured impact per build tuple	7-9 cycles	6-9 cycles	30-34 cycles	1-1.5 cycles



(a) Workload **A** (256 MiB ⋈ 4096 MiB)



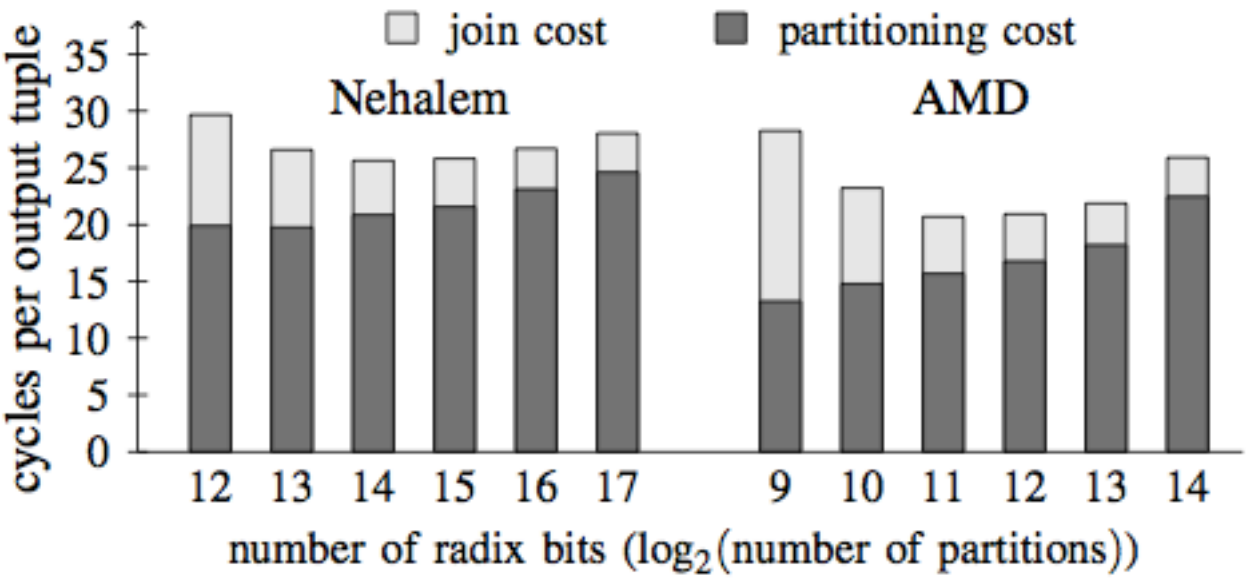
(b) Workload **B** (977 MiB ⋈ 977 MiB)

SPARCは弱いメモリモデルを
採用しているため同期コストが非常に低い
（＆キャッシュ効率の良し悪しの差）

※論文内Fig.14から引用

3. Main-memory Hash Joins on Multi-Cores CPUs

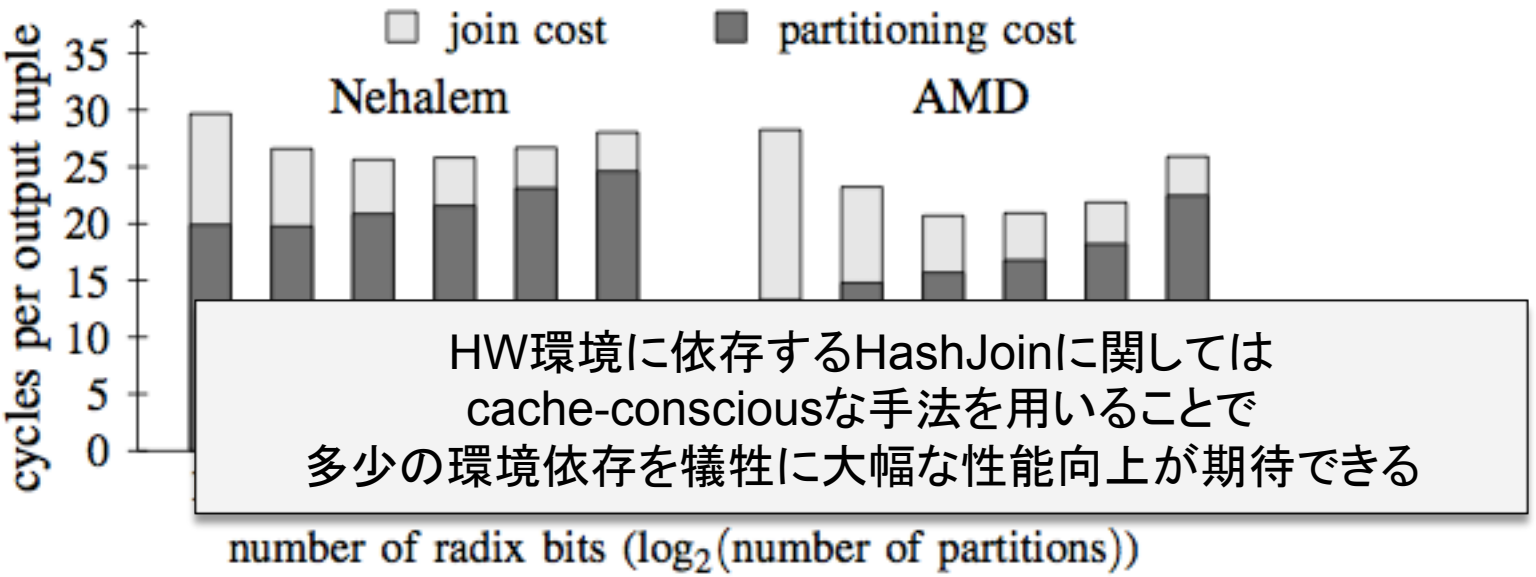
- Experimental
  - Radix JoinのPartition処理時間とKnobの関係は？
  - 多少の性能の上下はあるがknobの影響は小



※論文内Fig.9から引用

3. Main-memory Hash Joins on Multi-Cores CPUs

- Experimental
  - Radix JoinのPartition処理時間とKnobの関係は？
  - 多少の性能の上下はあるがknobの影響は小



HW環境に依存するHashJoinに関しては
cache-consciousな手法を用いることで
多少の環境依存を犠牲に大幅な性能向上が期待できる

※論文内Fig.9から引用