Scalable Frequent-Pattern Mining on Nonvolatile Memories

Yi Lin*, Po-Chun Huang, Duo Liu*, Liang liang*
*College of Computer Science, Chongqing University
Department of Computer Science and Engineering, Yuan Ze University
{liuduo}@cqu.edu.cn

19 Jan, 2016

Chiba, Japan

重慶大學 CHONGQING UNIVERSITY 元智大學 Yuan Ze University

Outline

- Introduction
- Motivation
- *PevFP-tree*
  - Link merge
  - Hash walk
- Evaluation
- Conclusion

Introduction

- Data mining and machine learning are the key technologies of data analytics, which reveal the hidden knowledge behind data

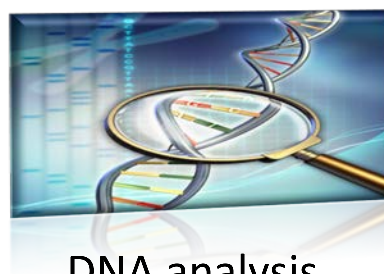
- Frequent pattern mining is a corner stone in data mining

Market basket analysis

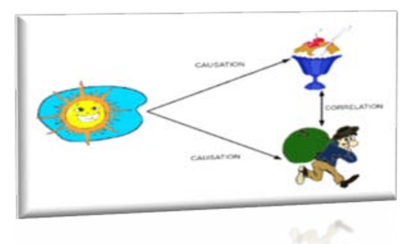
Classification

Text database analysis

Clustering



DNA analysis



Correlation or causality analysis

[1] H. Zhang etc., In-memory big data management and processing : A survey, *TKDE* 2015.

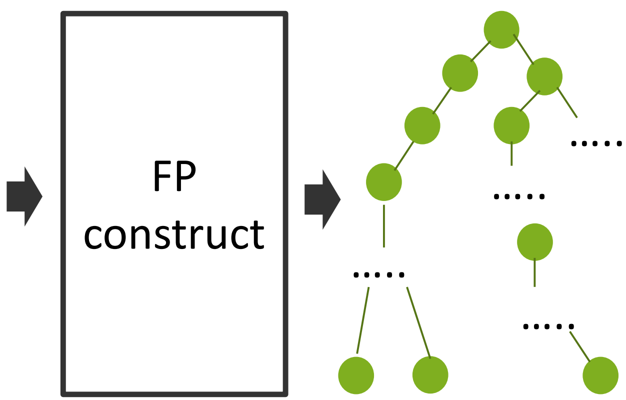
[2] Y. Wang etc., A survey of data mining softwares used for real projects, *OSSC* 2011.

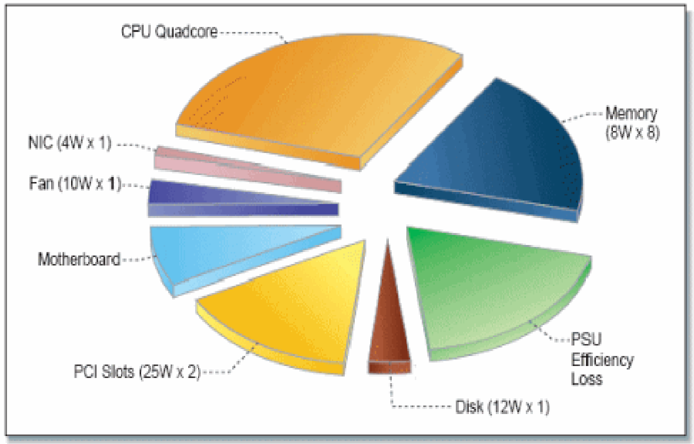
[3] J. Han etc., Mining frequent patterns without candidate generation, *SIGMOD* 2000.

Introduction

- Large DRAM is required for high-performance data mining.
- More DRAM implies higher energy consumption.
- We can use NVRAM to reduce the energy consumption.

TID	Items
1	*<a,b,c..f>*
2	*<b,d,e,g..t>*
……	……
10^{10}	*<c,d,f..z>*

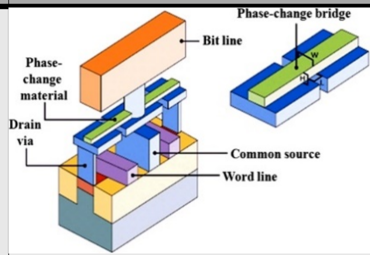
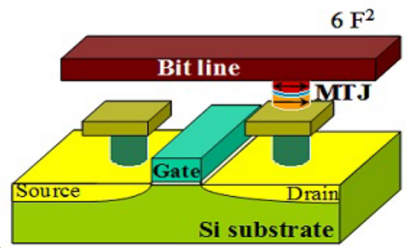
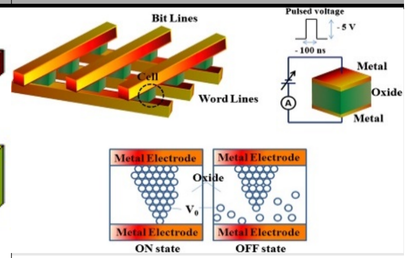




Intel, "The Problem of Power Consumption in Servers," available at https://software.intel.com/sites/default/files/m/d/4/1/d/8/power_consumption.pdf.

NVRAM

- Emerging Non-Volatile Memory (NVRAM)
  - Byte-addressable, high density, low standby power etc.
  - Near DRAM performance

	PCRAM	STT-MRAM	ReRAM
			
Cell Size	~4F2	14~6F2	4F2 &Stackable
Endurance	10E6~10E8	10E12~10E15	10E4~10E10
Read latency	100~300ns	~20ns	30ns~2us
Write latency	10~150us	~20ns	100ns~2us
Density	Medium/High	Low/medium	High
Cost	Medium	High	Low
Industrial Development	IBM&SK bynlx, Micron	SK hynlx & Toshiba Everspin	HP & SK Hynix Adesto, Crossbar

Outline

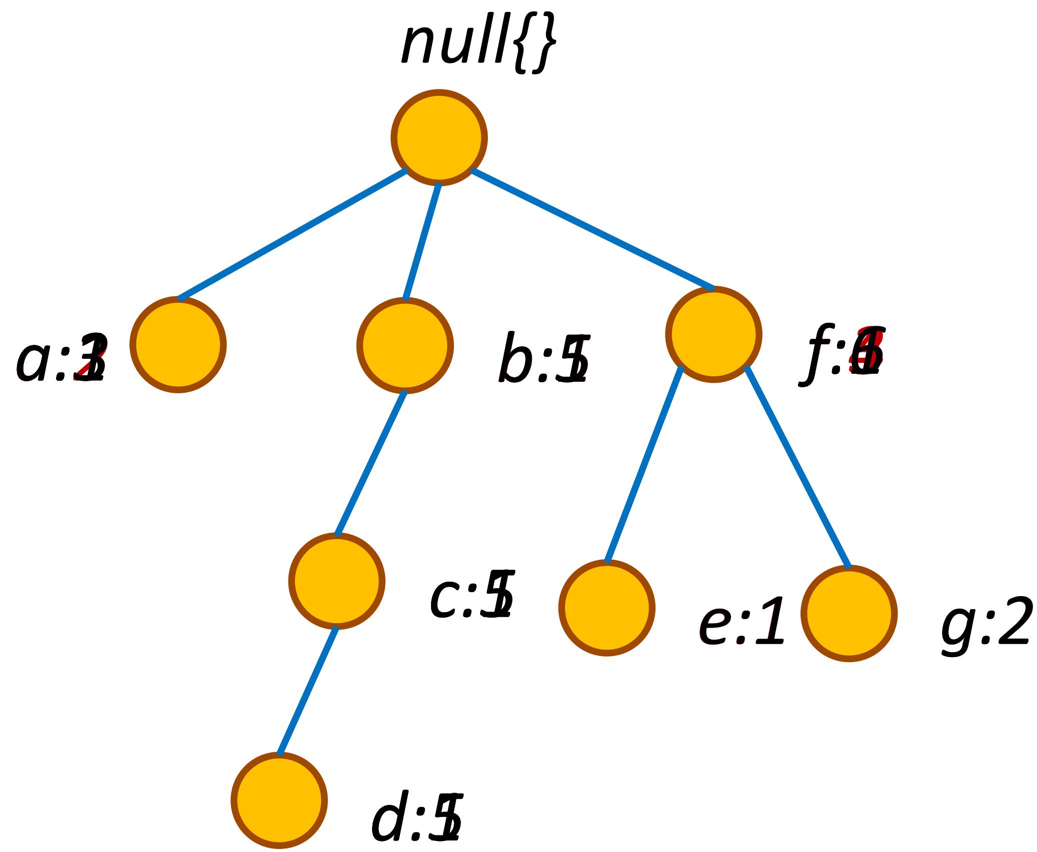
- Introduction
- Motivation
- *PevFP-tree*
  - Link merge
  - Hash walk
- Evaluation
- Conclusion

FP-tree

Consider the following database

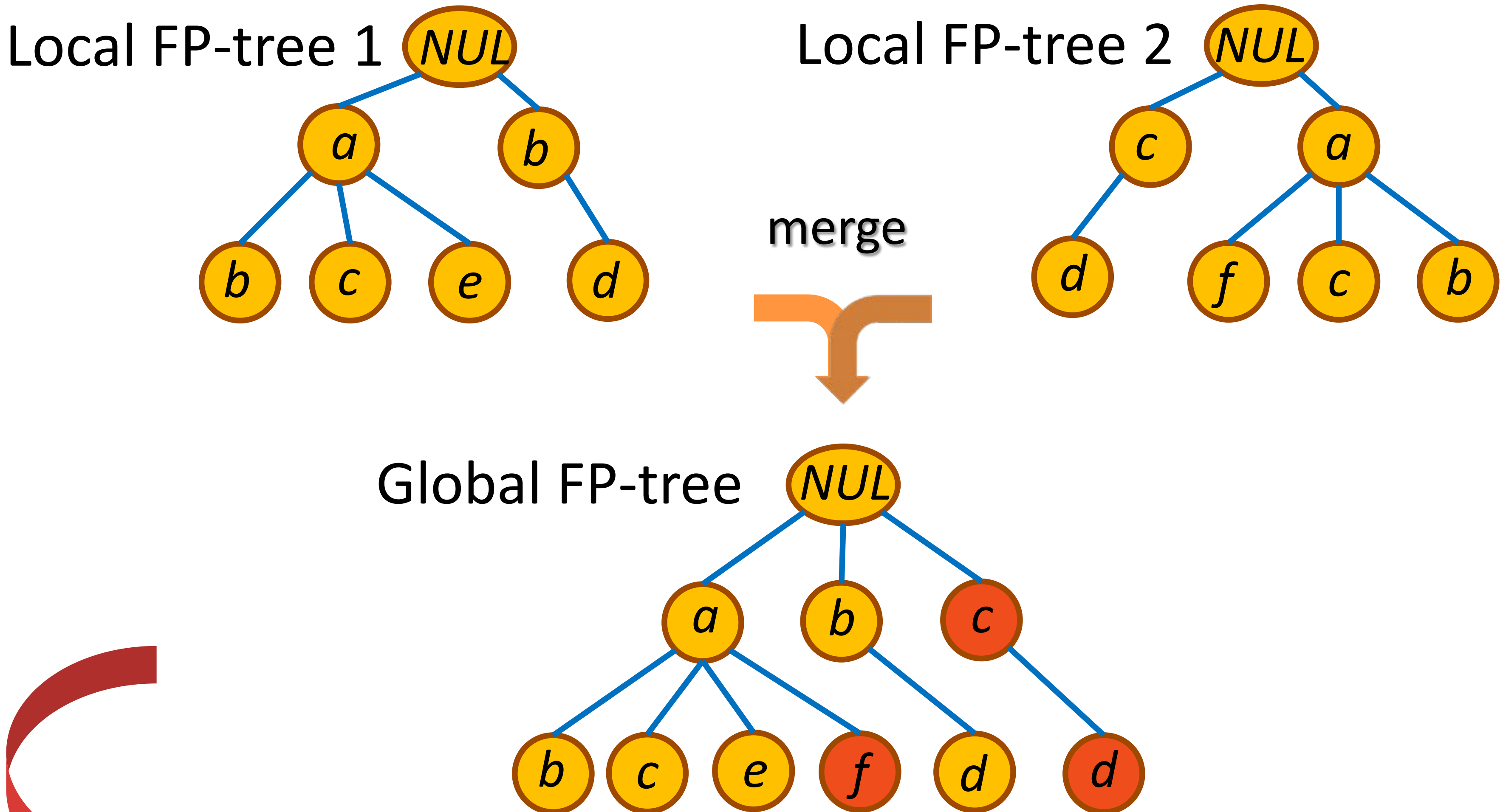
TID	Data items	TID	Data items
1	*<a>*	8	*<f>*
2	*<b,c,d>*	9	*<f>*
3	*<f,e>*	10	*<f>*
4	*<a>*	11	*<b,c,d>*
5	*<a>*	12	*<b,c,d>*
6	*<f,g>*	13	*<b,c,d>*
7	*<f,g>*	14	*<b,c,d>*

Completed FP-tree



This paper: enhance the scalability of in-memory frequent-pattern mining

Challenge 1: brute-force merging

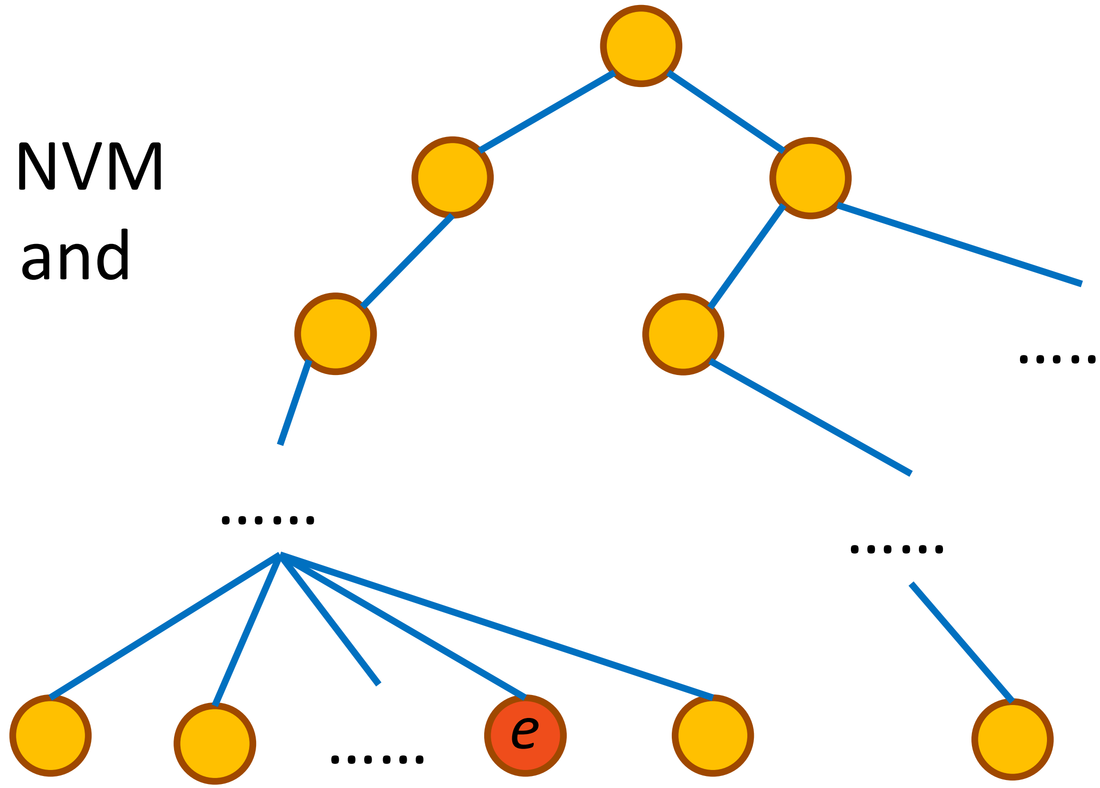


Link merge: reduce writes when merge multiple FP-trees

Challenge 2: large branching factor

- The lookup of the desired children of an FP-tree node could incur many NVM reads and become inefficient

- Too many reads on NVM will reduce lifetime and waste energy!

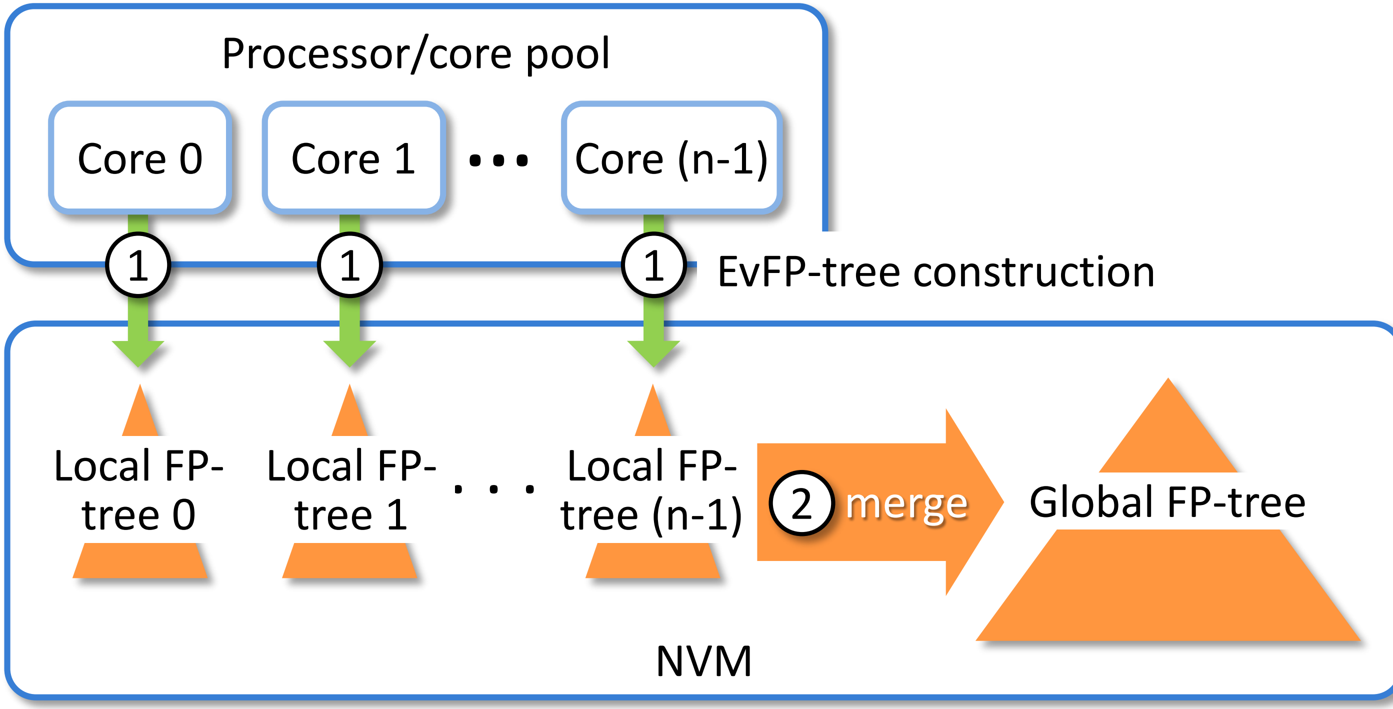


Hash walk: efficiently locate the desired FP-tree node

Outline

- Introduction
- Motivation
- *PevFP-tree*
  - Link merge
  - Hash walk
- Evaluation
- Conclusion

System architecture



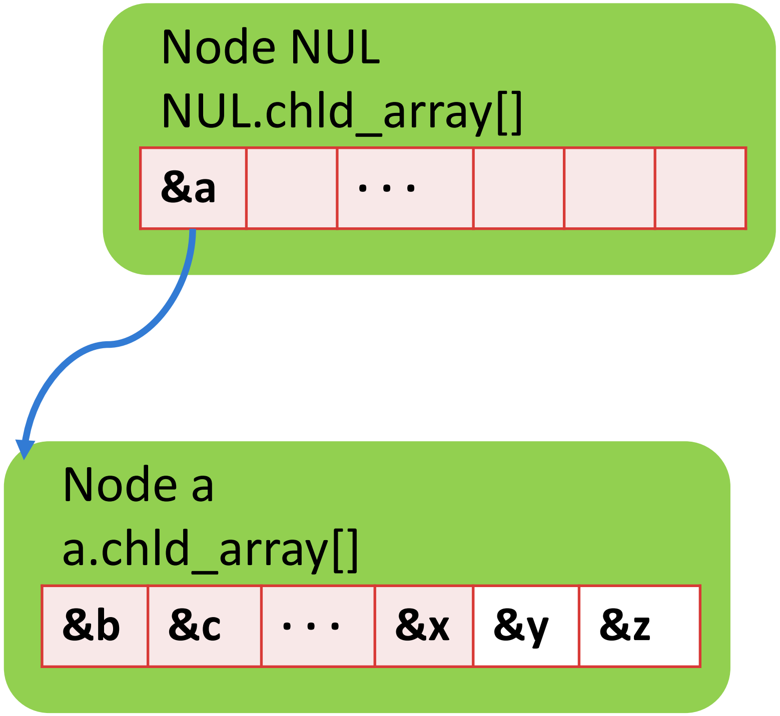
① Partition: partition the dataset into multiple parts, and each part will be processed by a core to construct a local FP-tree

② Merge: merge the local FP-tree into a global FP-tree

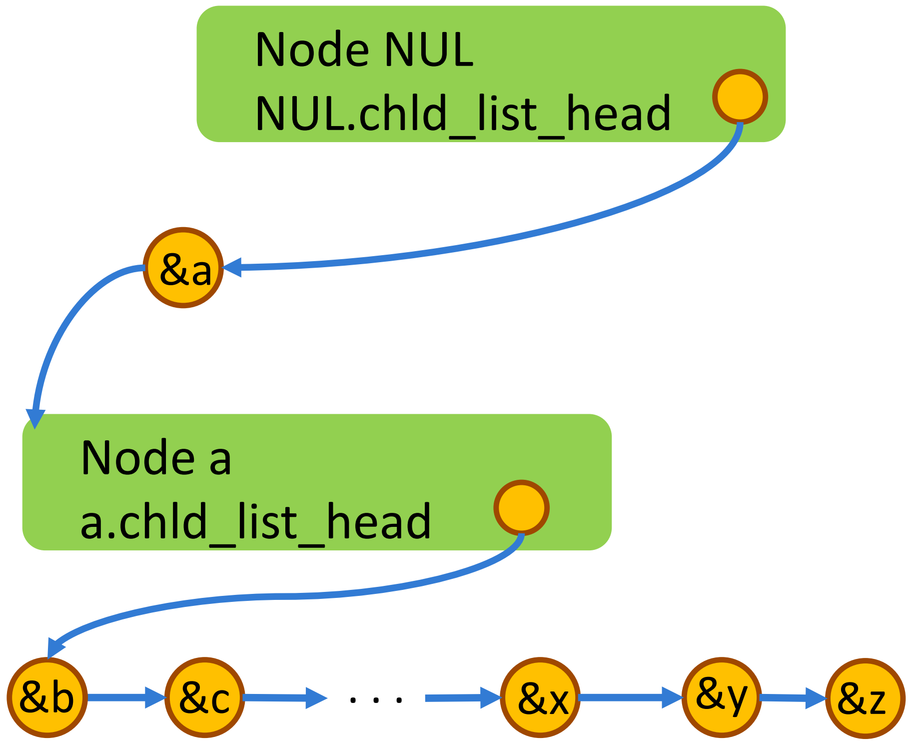
Outline

- Introduction
- Motivation
- *PevFP-tree*
  - Link merge
  - Hash walk
- Evaluation
- Conclusion

PevFP-tree: Link merge

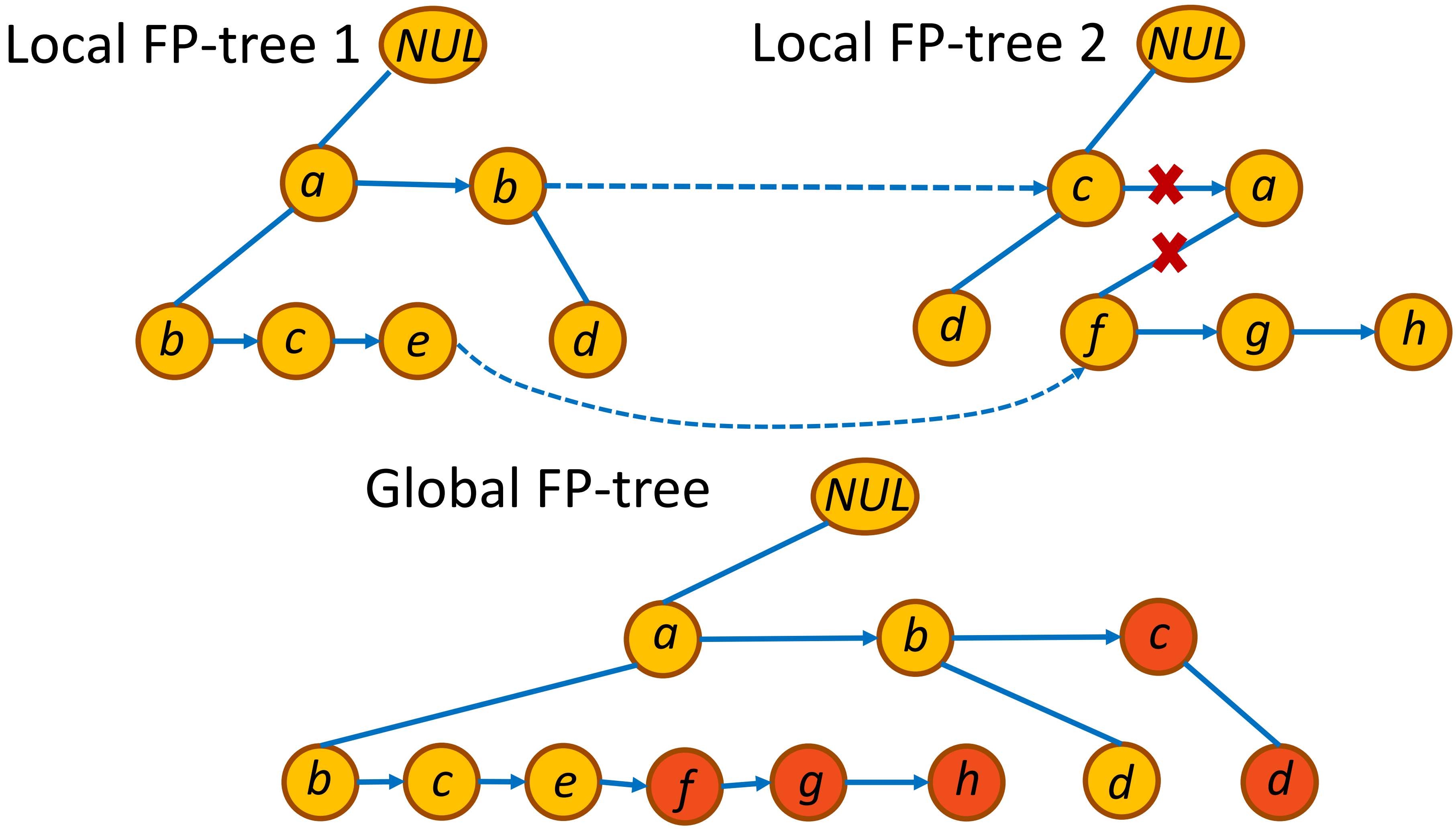


(a) Array of pointers to child nodes



(b) Left-sibling linked list of pointers to child nodes

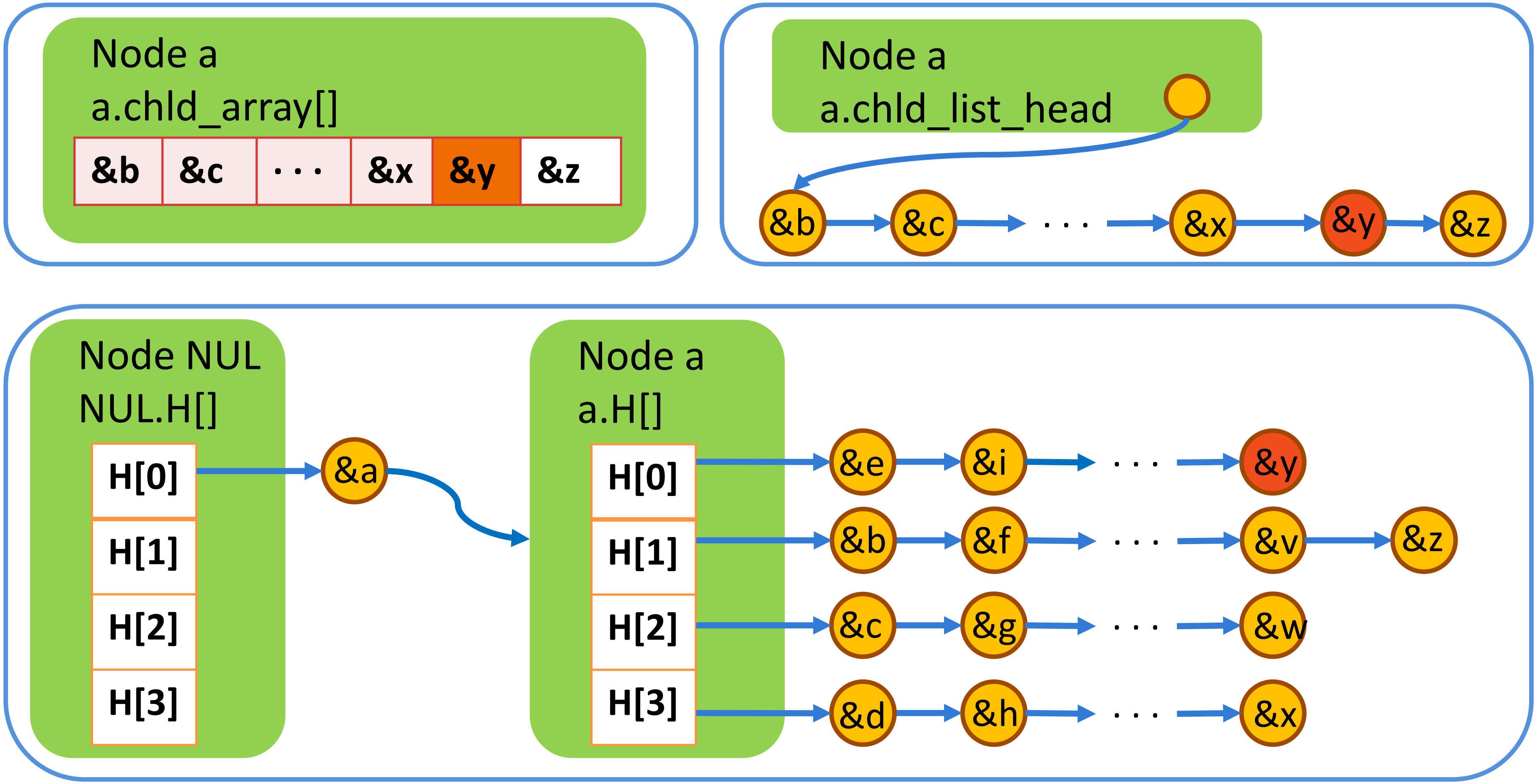
PevFP-tree: Link merge



Outline

- Introduction
- Motivation
- *PevFP-tree*
  - Link merge
  - Hash walk
- Evaluation
- Conclusion

PevFP-tree: Hash walk



- Use hash table to reduce read operations

Outline

- Introduction
- Motivation
- *PevFP-tree*
    - Link merge
    - Hash walk
- Evaluation
- Conclusion

Evaluation: experimental setup

	Description
OS	RedHat 6.0 with kernel version 2.6.3
Read latency	6.82ns
Write latency	152.20ns
Reset latency	12.20ns
Cache	32 KB, 4 way associative, LRU

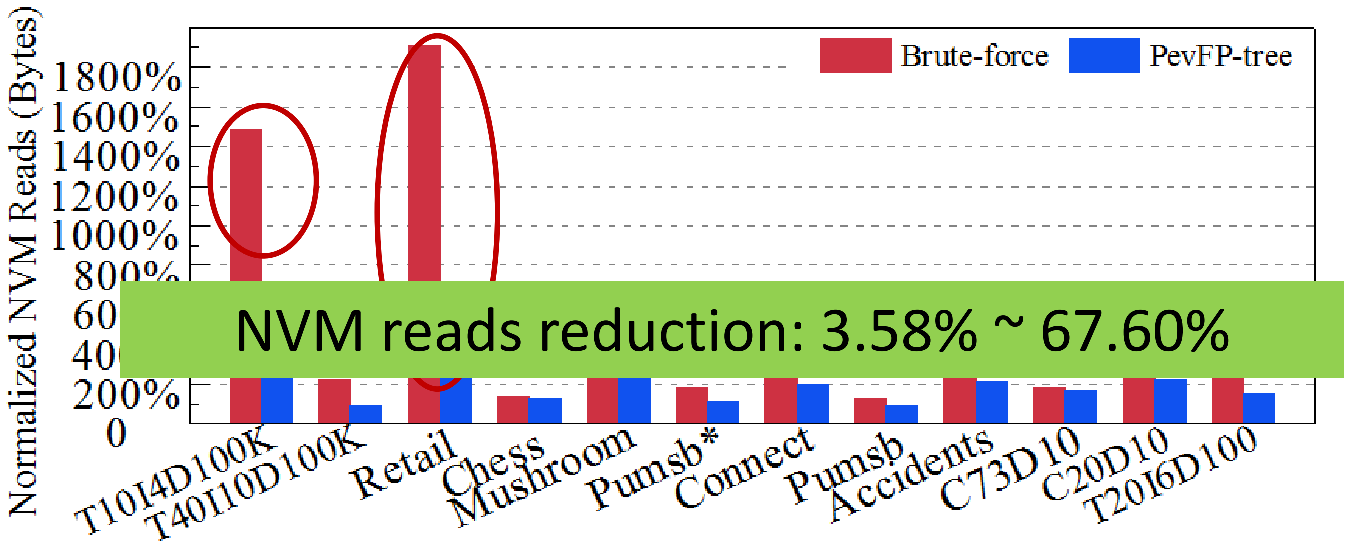
Evaluation: datasets

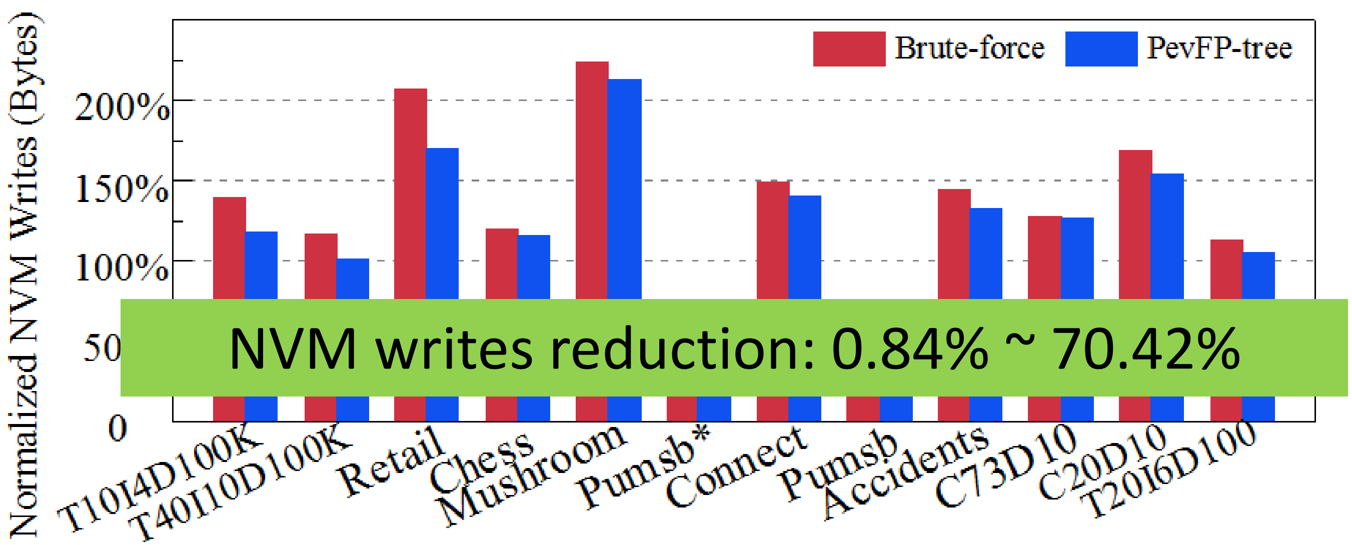
Dataset	# Trans.	# Items	Max. Trans. Leng.	Avg. Trans. Leng.	Size (MB)
T10I4D100K	100,000	870	29	10.10	4.00
T40I10D100K	100,000	942	77	39.61	15.5
retail	88,162	16,470	76	10.31	4.2
chess	3,196	75	37	37.00	0.34
mushroom	8,124	119	23	23.00	0.57
pumsb*	49,046	2,088	63	50.48	11.3
connect	67,557	129	43	43.00	9.3
pumsb	49,046	2,113	74	74.00	16.7
accidents	340,183	468	51	33.81	35.5
C73D10	10,000	1,592	73	73	3.2
C20D10	2,000	1,592	20	20	0.16
T20I6D100	99,922	893	47	19.90	7.8

Experimental: metrics

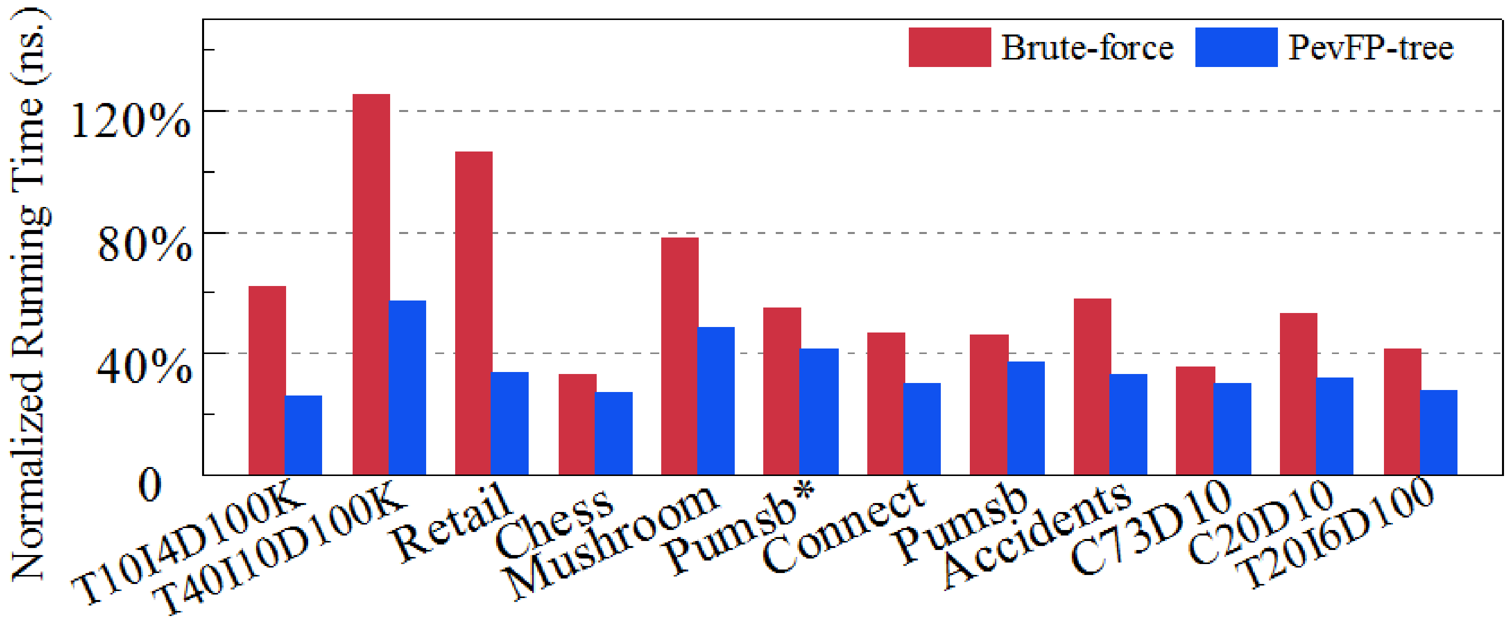
- # of NVM reads
- # of NVM writes
- Total time for global FP-tree construction
- Total running time with different parallelism degrees

Experiment Results



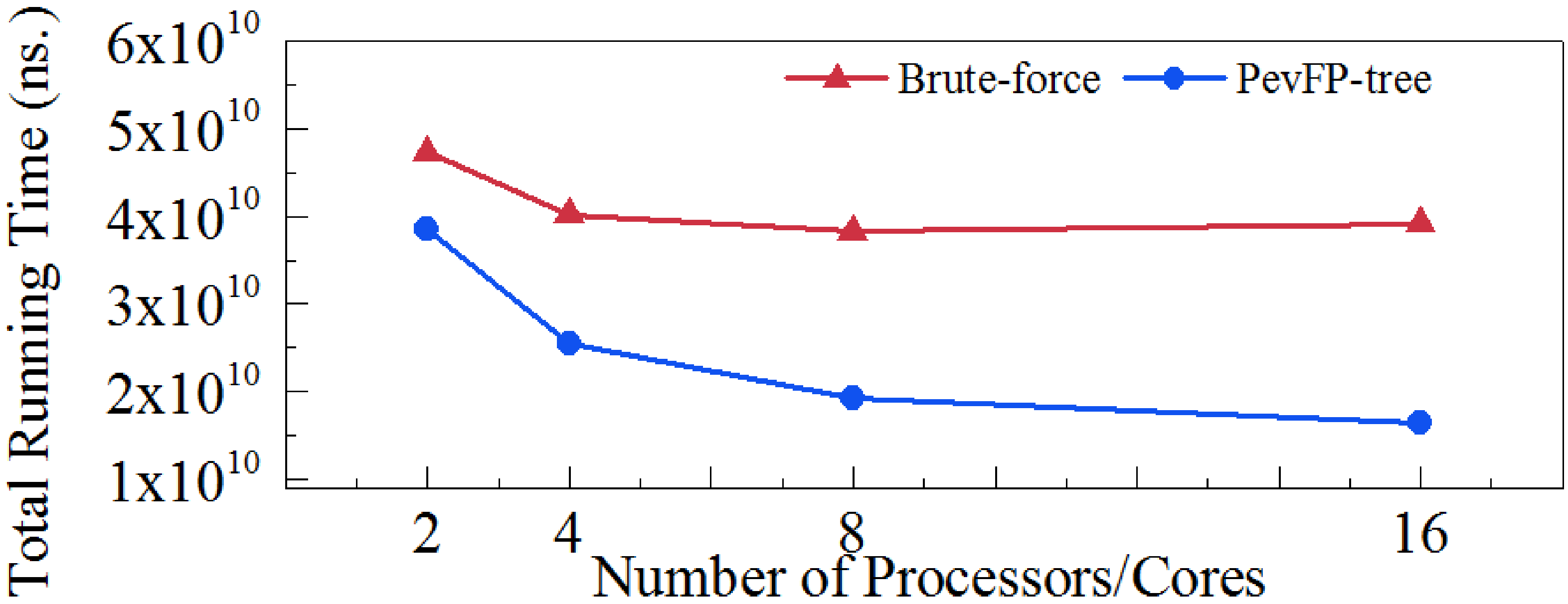


Experiment Results



- Compare to Brute-force approach, PevFP-tree improves running time by 20% on average.

Experiment Results



- The total running time could be reduced with the number of cores increased through both PevFP-tree and brute-force

Outline

- Introduction
- Motivation
- *PevFP-tree*
  - Link merge
  - Hash walk
- Evaluation
- Conclusion

Conclusion

- We proposed *PevFP-tree* to enhance the scalability of in-memory frequent-pattern mining on NVMs

- *Link Merge*: alleviate the overheads to merge multiple local FP-trees into the global FP-tree

- *Hash Walk*: efficiently locate the FP-tree node of the desired data item

- Results show that the proposed approach achieves significant performance improvement.

Thank You!

Q&A?