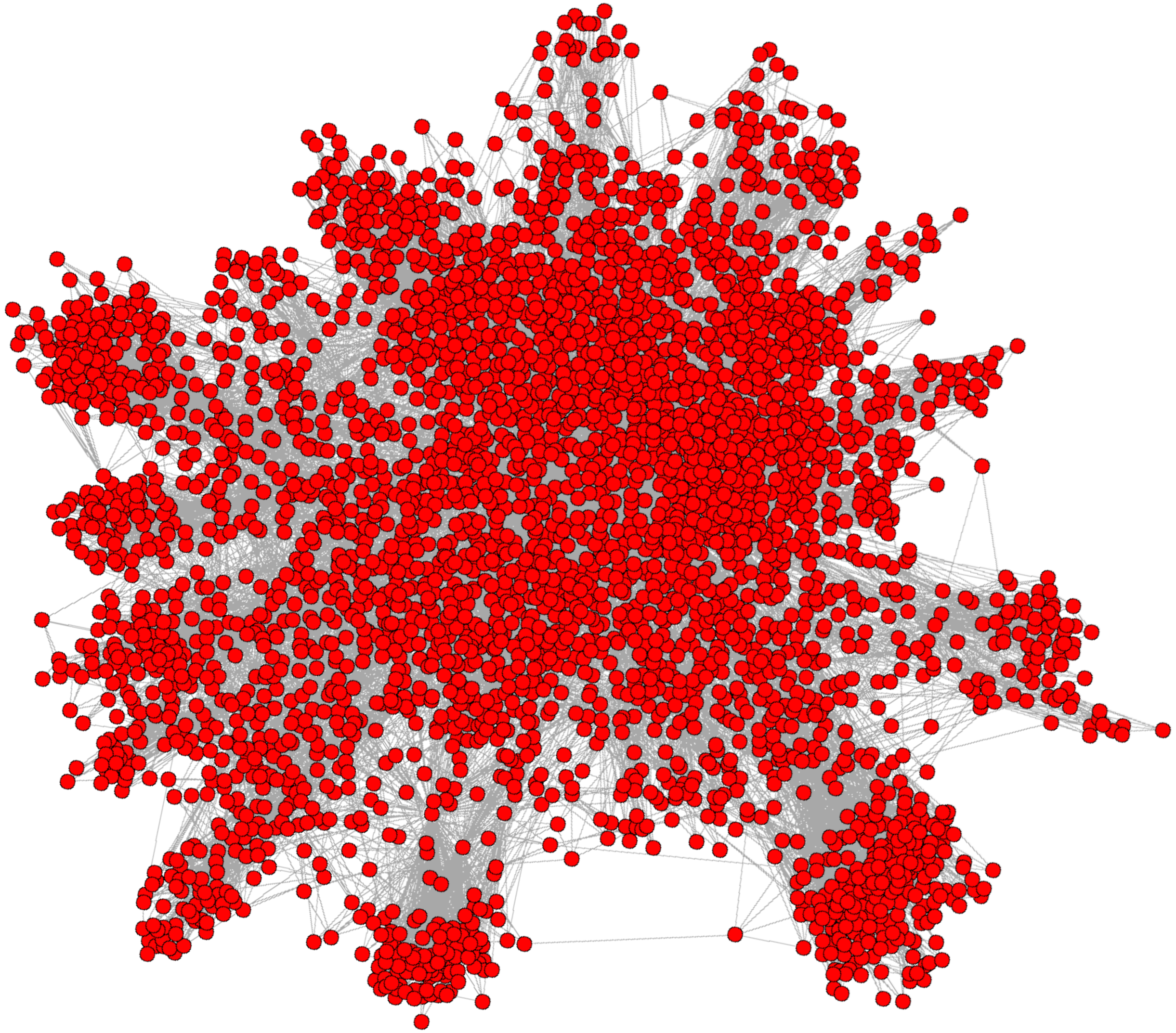
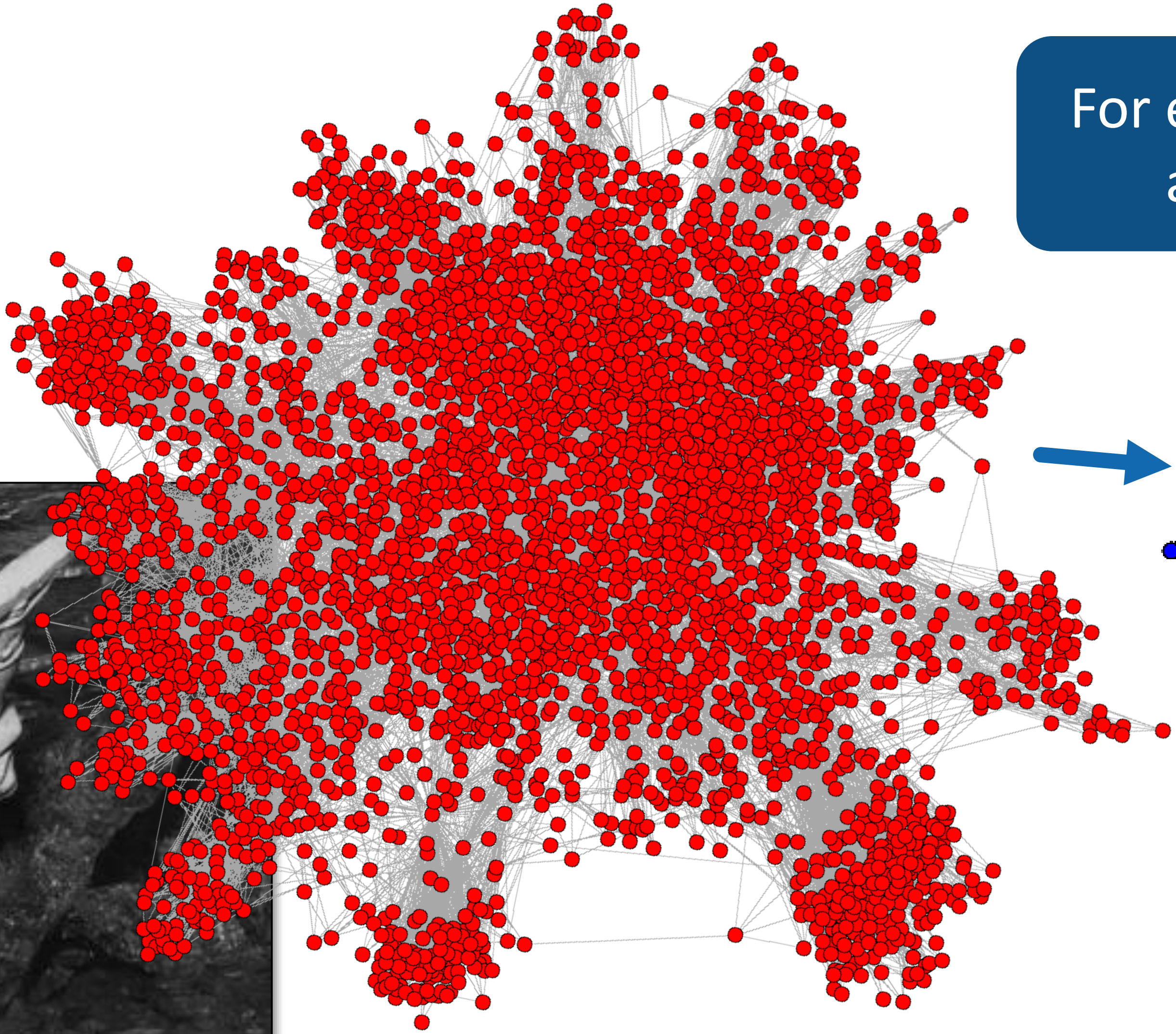
SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems

Maciej Besta, Raghavendra Kanakagiri, Grzegorz Kwasniewski, Rachata Ausavarungnirun, Jakub Beránek, Konstantinos Kanellopoulos, Kacper Janda, Zur Vonarburg-Shmaria, Lukas Gianinazzi, Ioana Stefan, Juan Gómez Luna, Marcin Copik, Lukas Kapp-Schwoerer, Salvatore Di Girolamo, Marek Konieczny, Onur Mutlu, Torsten Hoefler

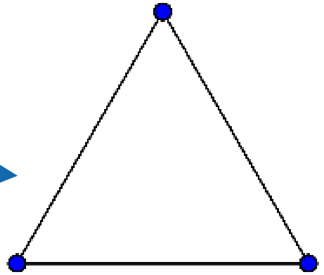
Graph Mining



Graph Mining

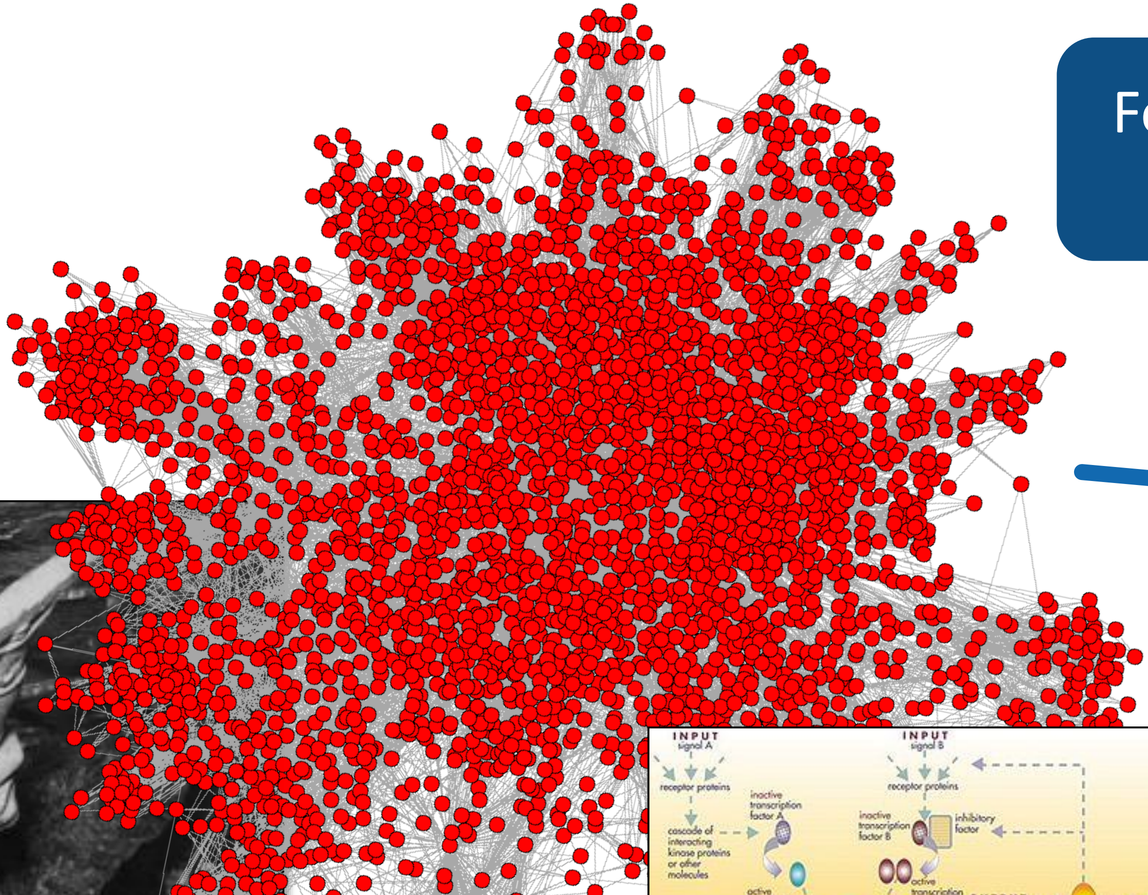


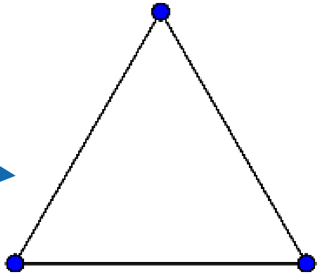
For example, listing all k-cliques

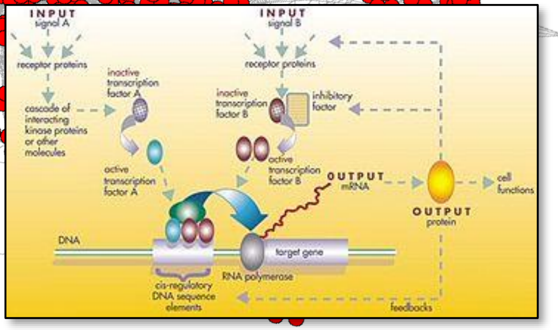


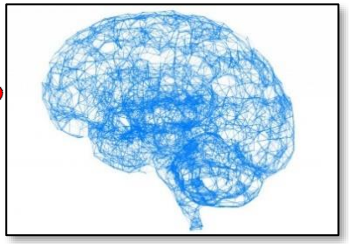
Graph Mining

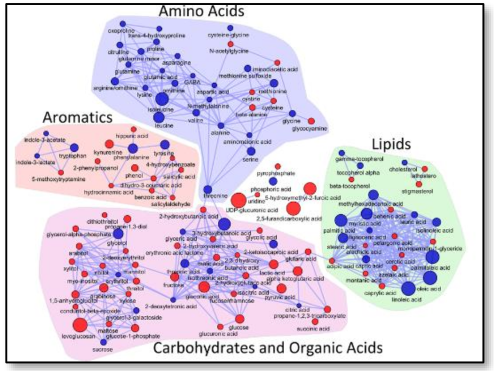
For example, listing all k-cliques







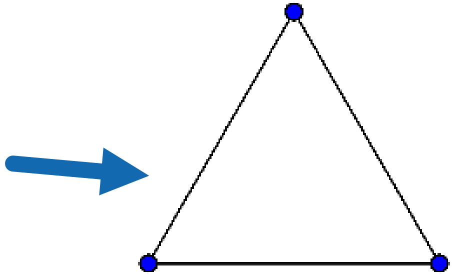


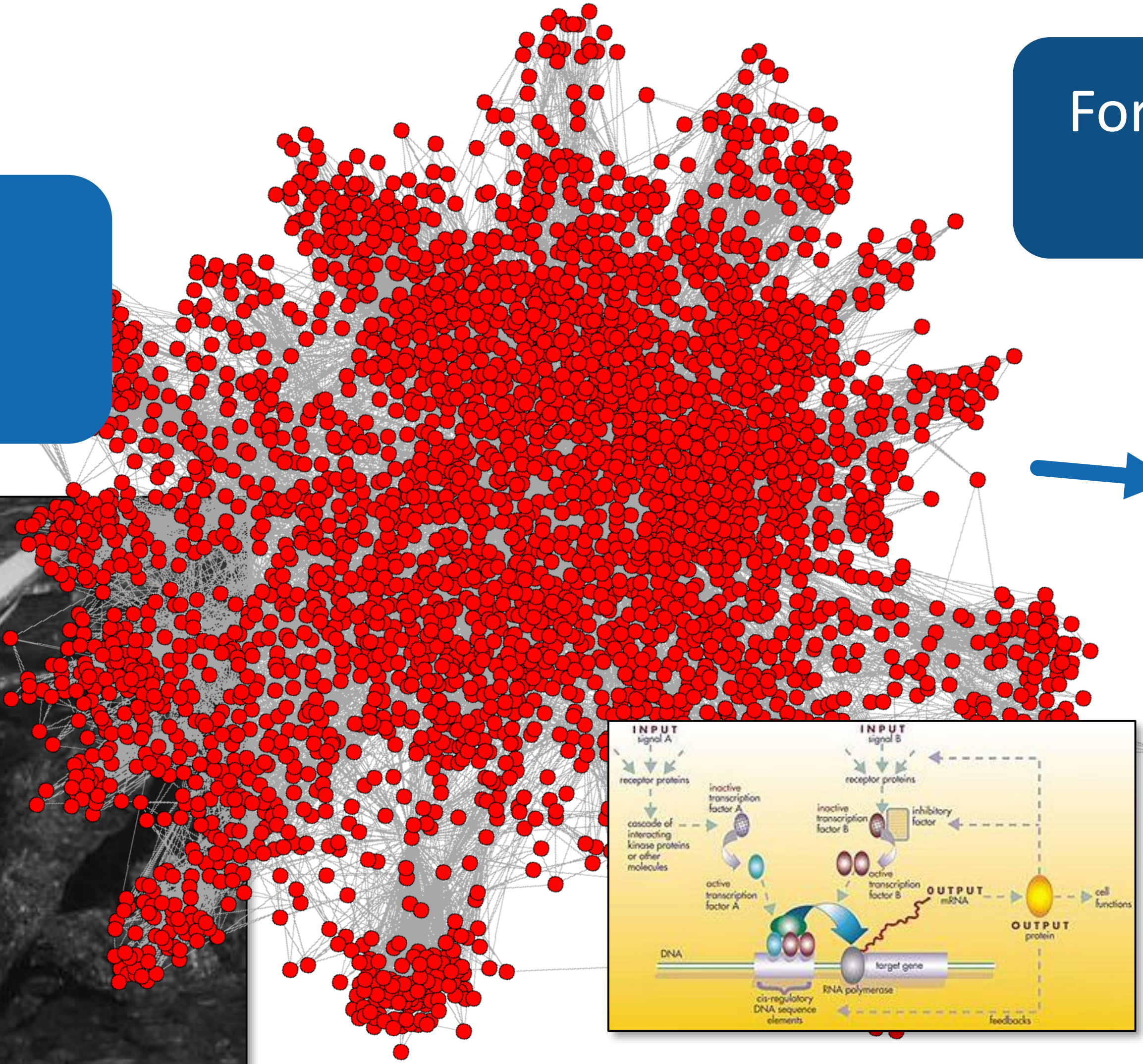


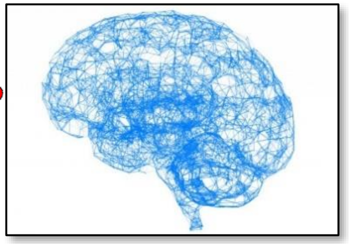
Graph Mining

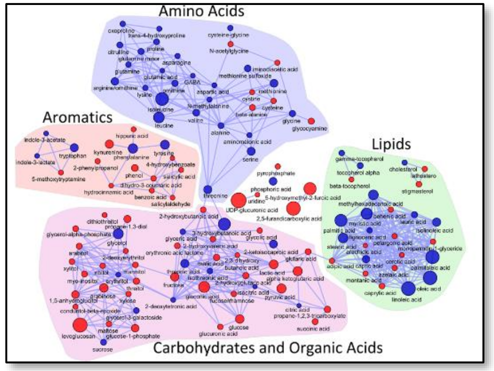
Challenges?

For example, listing all k-cliques









Graph Mining: Challenges

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

Complex algorithm structure, deeply recursive, no notion of iterations

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

Complex algorithm structure, deeply recursive, no notion of iterations

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

Complex algorithm structure, deeply recursive, no notion of iterations

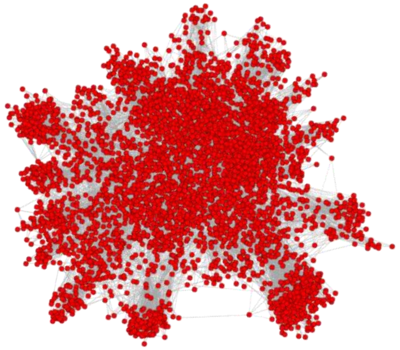
Non-straightforward parallelism, complicated memory access patterns

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

```
while not done
  for all vertices v:
    send updates over outgoing edges of v
  for all vertices v:
    apply updates from inbound edges of v
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing



Complex algorithm structure, deeply recursive, no notion of iterations

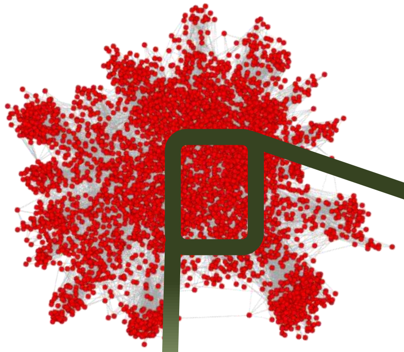
Non-straightforward parallelism, complicated memory access patterns

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

```
while not done
  for all vertices v:
    send updates over outgoing edges of v
  for all vertices v:
    apply updates from inbound edges of v
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing



Complex algorithm structure, deeply recursive, no notion of iterations

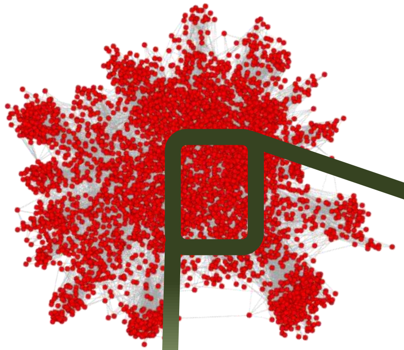
Non-straightforward parallelism, complicated memory access patterns

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

```
while not done
  for all vertices v:
    send updates over outgoing edges of v
  for all vertices v:
    apply updates from inbound edges of v
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

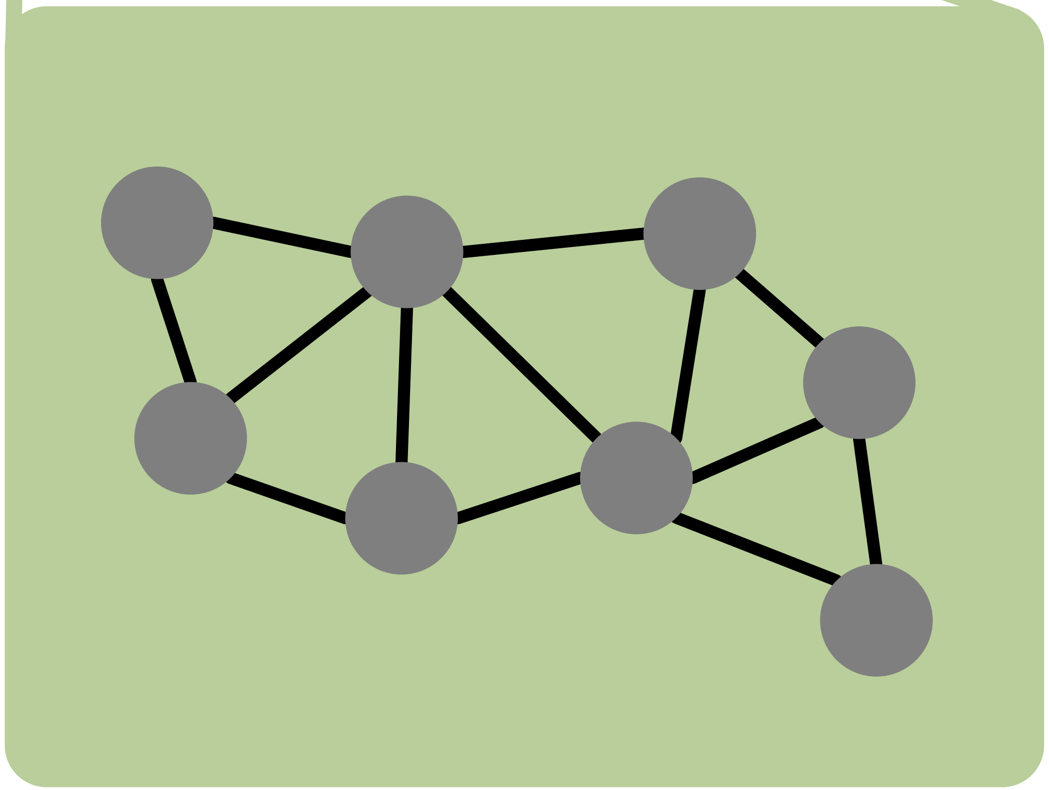


Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

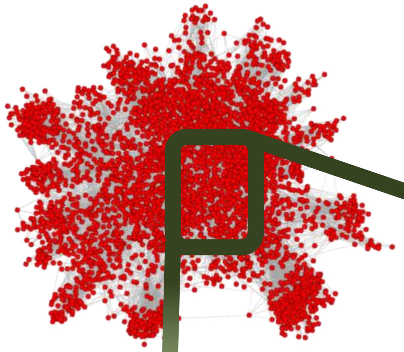
```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

```
while not done
  for all vertices v:
    send updates over outgoing edges of v
  for all vertices v:
    apply updates from inbound edges of v
```



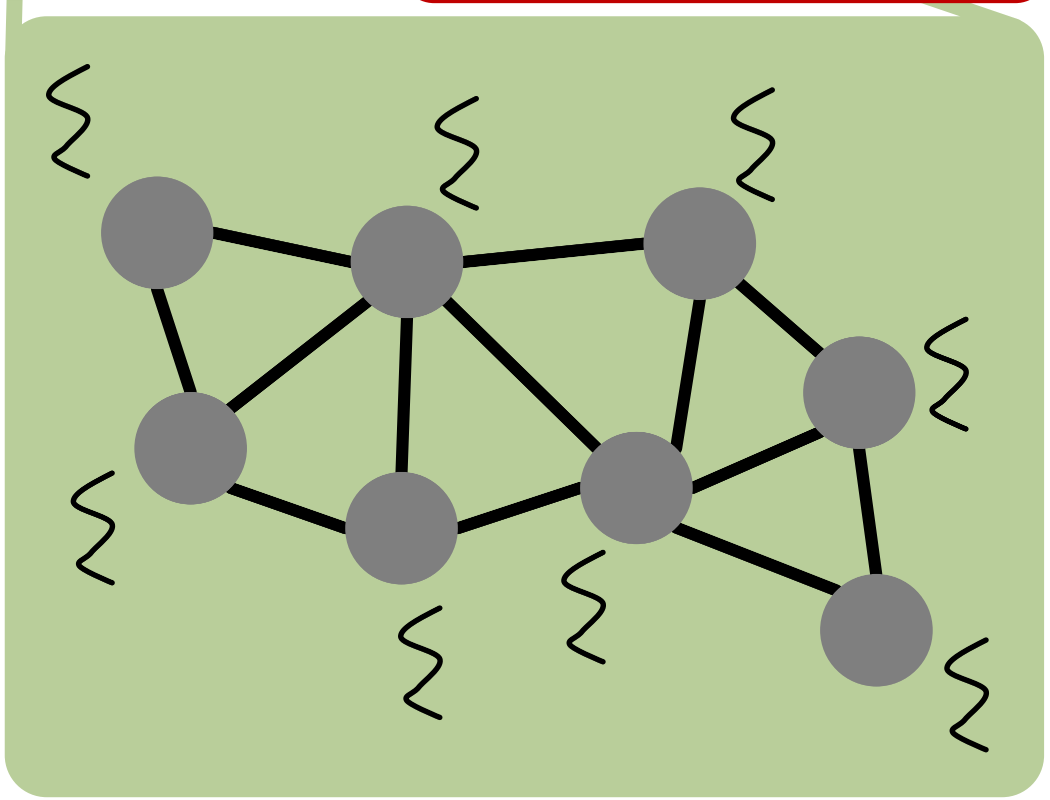
Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing



Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

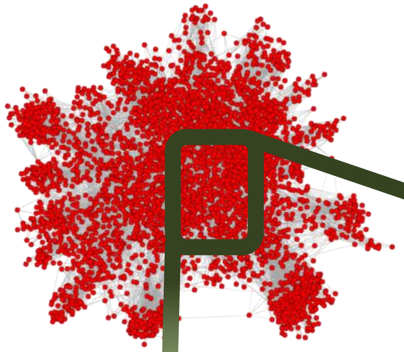


```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

```
while not done
  for all vertices v:
    send updates over outgoing edges of v
  for all vertices v:
    apply updates from inbound edges of v
```

Graph Mining: Challenges

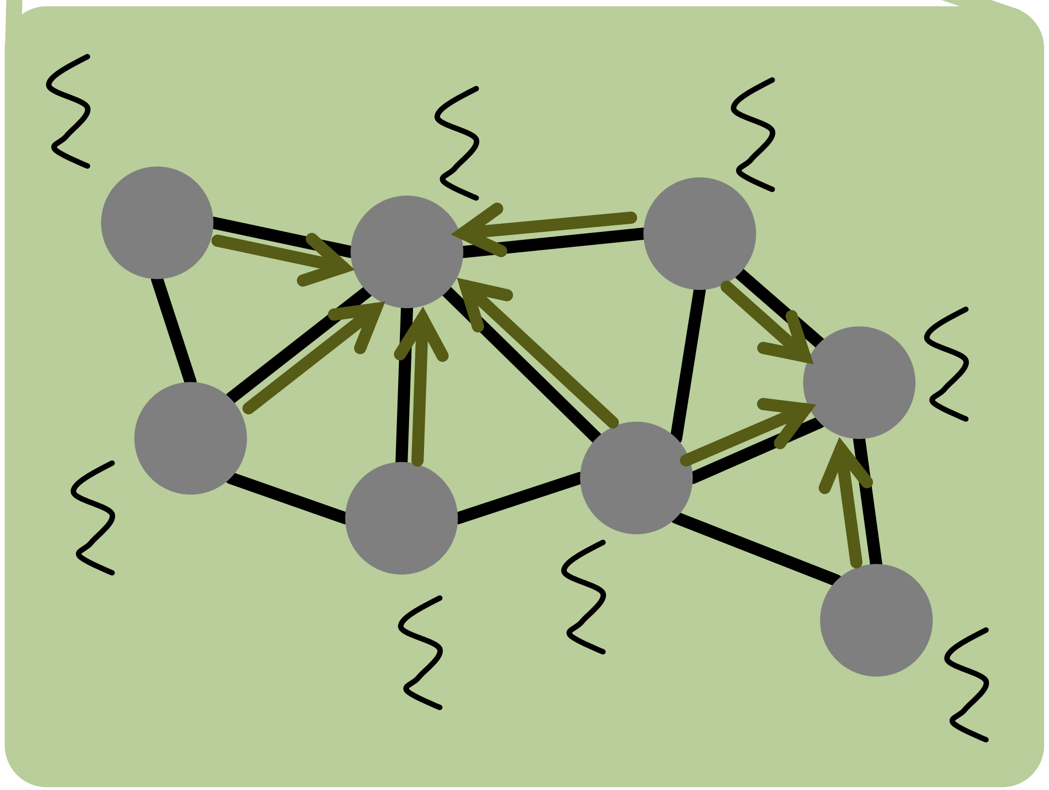
Example: the Bron-Kerbosch algorithm for maximal clique listing



Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

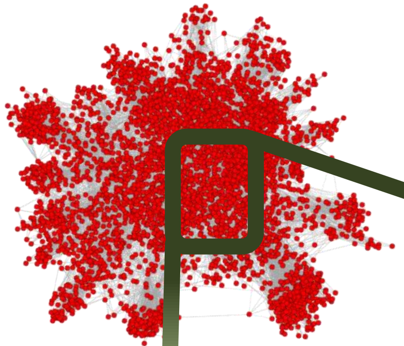
```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```



```
while not done
  for all vertices v:
    send updates over outgoing edges of v
  for all vertices v:
    apply updates from inbound edges of v
```

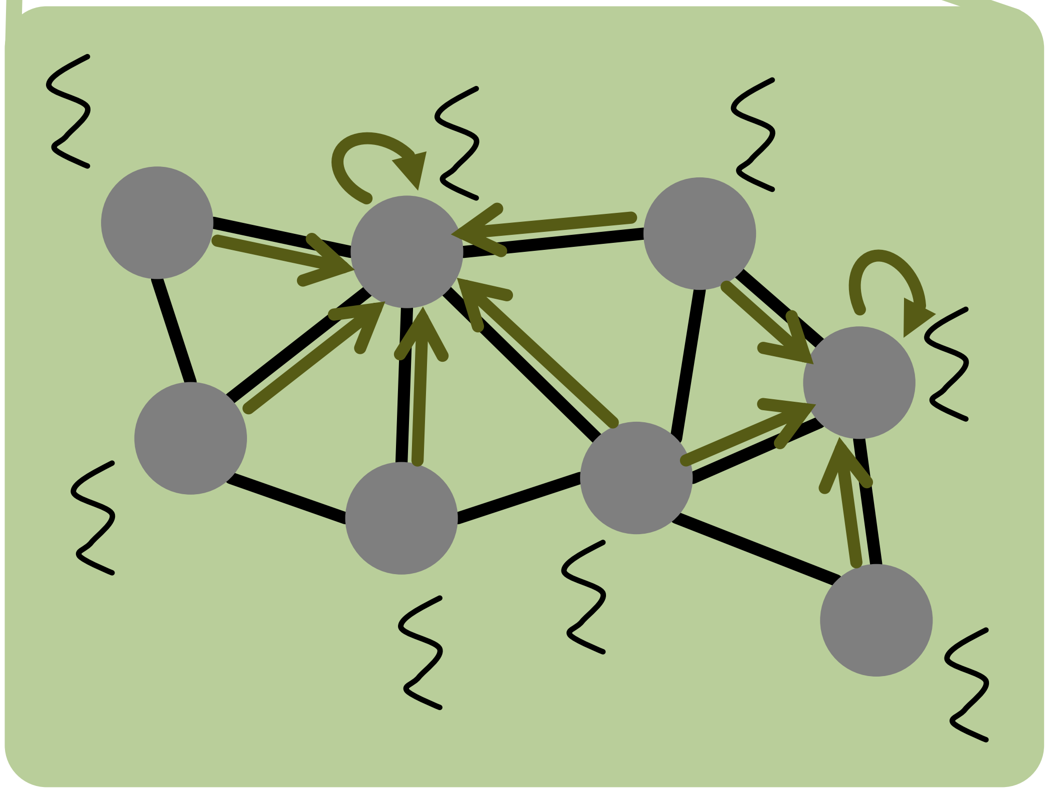
Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing



Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

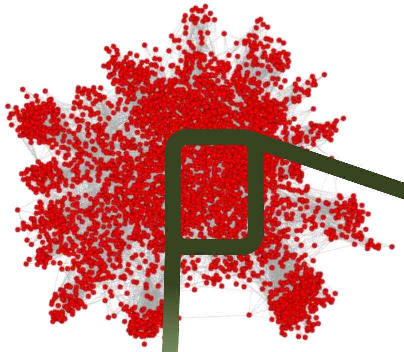


```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

```
while not done
  for all vertices v:
    send updates over outgoing edges of v
  for all vertices v:
    apply updates from inbound edges of v
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing



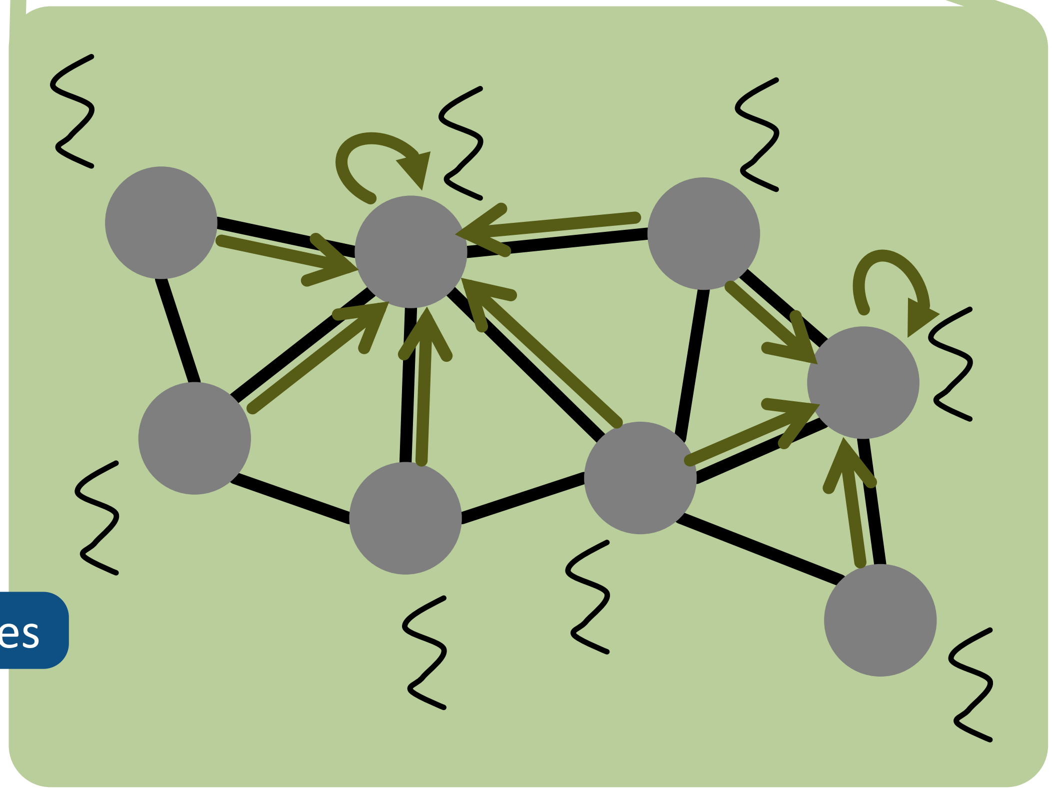
Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

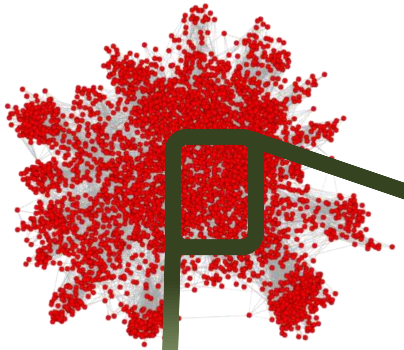
```
while not done
  for all vertices v:
    send updates over outgoing edges of v
  for all vertices v:
    apply updates from inbound edges of v
```

...Repeat several times



Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing



Complex algorithm structure, deeply recursive, no notion of iterations

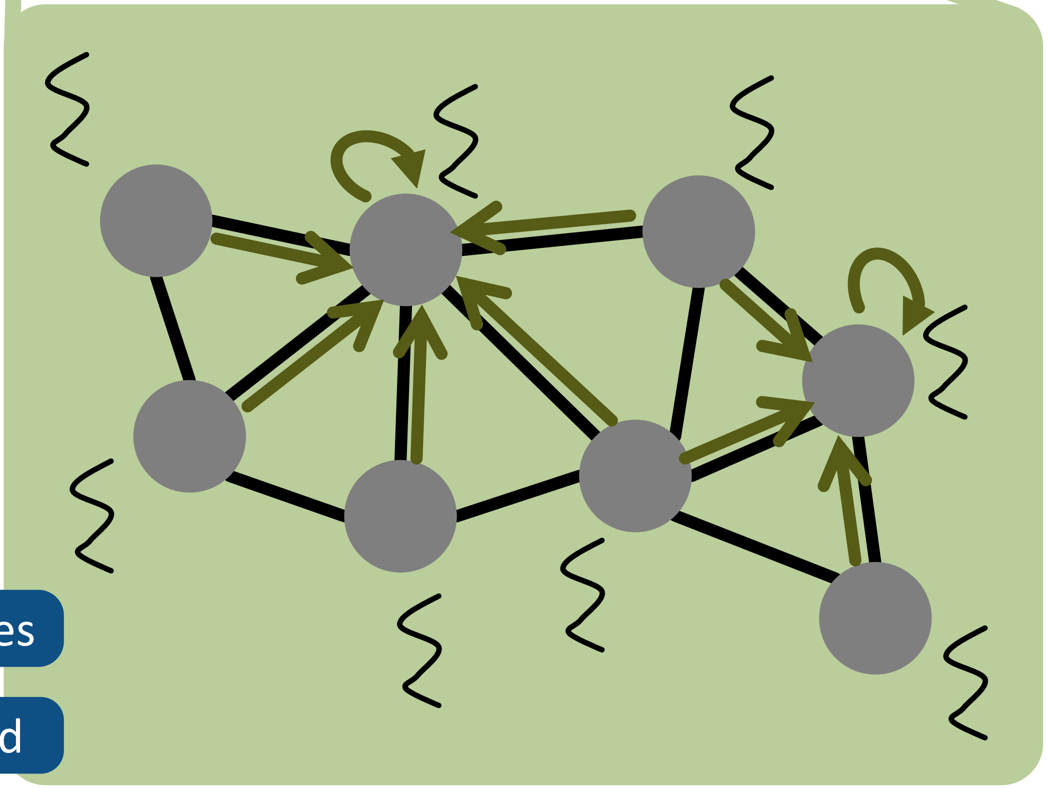
Non-straightforward parallelism, complicated memory access patterns

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

```
while not done
  for all vertices v:
    send updates over outgoing edges of v
  for all vertices v:
    apply updates from inbound edges of v
```

...Repeat several times

Not very complicated



Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

```
algorithm BronKerbosch(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

Many other algorithms with similar properties

```
algorithm BronKerbosch(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

Graph Mining: Challenges

k-clique listing

Clustering

Dense subgraph discovery

Vertex orderings

Subgraph isomorphism

Vertex similarity

Frequent subgraph mining

Link prediction

Example: the Bron-Kerbosch algorithm for maximal clique listing

Many other algorithms with similar properties

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Graph Mining: Challenges

k-clique listing

Clustering

Dense subgraph discovery

Vertex orderings

Subgraph isomorphism

Vertex similarity

Frequent subgraph mining

Link prediction

Example: the Bron-Kerbosch algorithm for maximal clique listing

Many other algorithms with similar properties

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

Hardware getting not just massively parallel but also massively heterogeneous

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Graph Mining: Challenges

k-clique listing

Clustering

Dense subgraph discovery

Vertex orderings

Subgraph isomorphism

Vertex similarity

Frequent subgraph mining

Link prediction

Example: the Bron-Kerbosch algorithm for maximal clique listing

Many other algorithms with similar properties

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

Hardware getting not just massively parallel but also massively heterogeneous

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

Many other algorithms with similar properties

k-clique listing

Clustering

Dense subgraph discovery

Subgraph isomorphism

Vertex orderings

Vertex similarity

Frequent subgraph mining

Link prediction

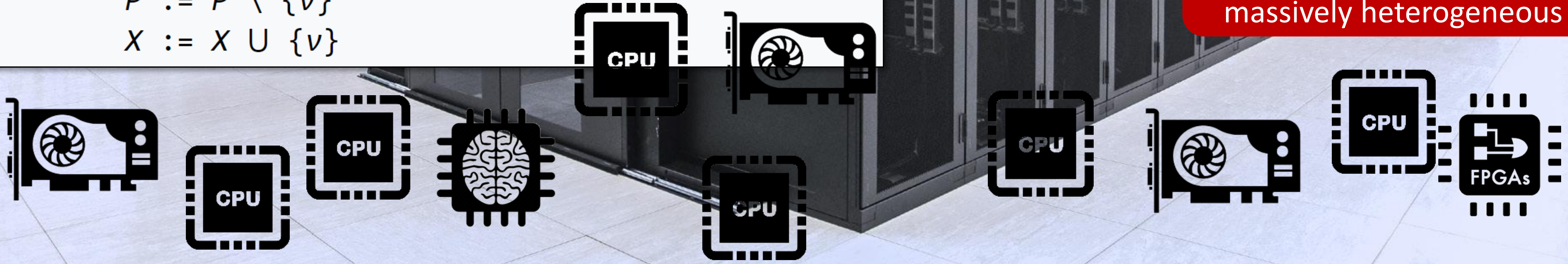
Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

Hardware getting not just massively parallel but also massively heterogeneous

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```



Graph Mining: Challenges

k-clique listing

Clustering

Dense subgraph discovery

Subgraph isomorphism

Vertex orderings

Vertex similarity

Link prediction

Frequent subgraph mining

Example: the Bron-Kerbosch algorithm for maximal clique listing

Many other algorithms with similar properties

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

Hardware getting not just massively parallel but also massively heterogeneous

```
algorithm BronKerbosch(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch(R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Goal: a paradigm that enables high-performance graph mining algorithms, that is…

Gr

Ex
Bro
alg
maximal
clique listing

properties

Clustering

Vertex
rderings

Link prediction

Frequent subgraph mining

Dense subgraph discovery

Vertex similarity

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

Hardware getting not just massively parallel but also massively heterogeneous

```
algorithm BronKerbosch(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

CPU
CPU
CPU
CPU
CPU
CPU
FPGAs

Gr

Goal: a paradigm that enables high-performance graph mining algorithms, that is…

…general (applicable to many problems)

Ex
Bro
al
maximal clique listing

properties

Link prediction

Clustering

Vertex orderings

Dense subgraph discovery

Frequent subgraph mining

Vertex similarity

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

Hardware getting not just massively parallel but also massively heterogeneous

```
algorithm BronKerbosch(R, P,
    if P and X are both empty th
        report R as a maximal cl
    choose a pivot vertex u in P
    for each vertex v in P \ N(u) do
        BronKerbosch(R U {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X U {v}
```

CPU CPU CPU CPU CPU CPU FPGAs

Goal: a paradigm that enables high-performance graph mining algorithms, that is...

...general (applicable to many problems)

... theoretically efficient (low work) for more performance

Gr...

Ex...
Br...
al...
maximal
clique listing

properties

Clustering

Vertex
...rderings

Link prediction

Frequent
subgraph
mining

Dense
subgraph
discovery

Vertex
similarity

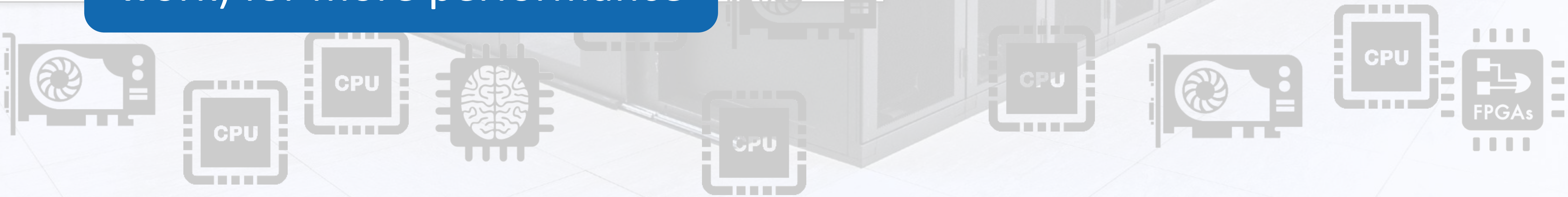
Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

Hardware getting not just massively parallel but also massively heterogeneous

```
algorithm BronKerbosch(R, P, ...
    if P and X are both empty th...
        report R as a maximal cl...
    choose a pivot vertex u in P ...
    f... each vertex v in P \ N(u) do
                              ...X ∩ N(v))
```



Goal: a paradigm that enables high-performance graph mining algorithms, that is...

...general (applicable to many problems)

... theoretically efficient (low work) for more performance

... easily harnesses the potential of diverse hardware

Clustering

Vertex orderings

Dense subgraph discovery

Vertex similarity

Frequent subgraph mining

Link prediction

properties

maximal clique listing

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

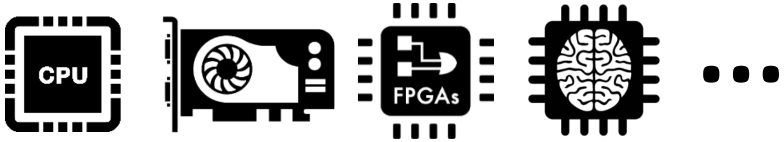
Many algorithms are NP-complete or even EXPTIME

Hardware getting not just ... but also ...eneous

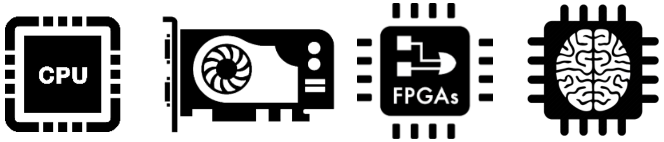
```
algorithm BronKerbosch(R, P, ...
    if P and X are both empty th...
        report R as a maximal cl...
    choose a pivot vertex u in P ...
    for each vertex v in P \ N(u) do
```

...What Hardware?

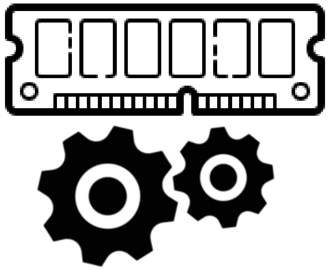
...What Hardware?



...What Hardware?



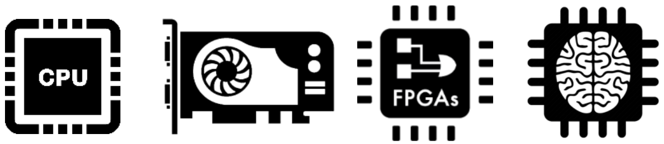
...



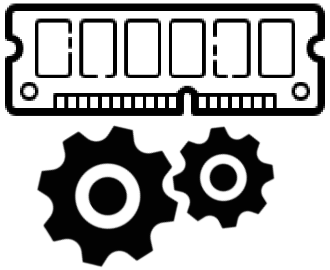
Processing in Memory (PIM) [1]

[1] O. Mutlu et al., A modern primer on processing in memory. 2021.

...What Hardware?

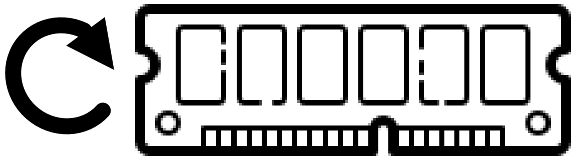


...



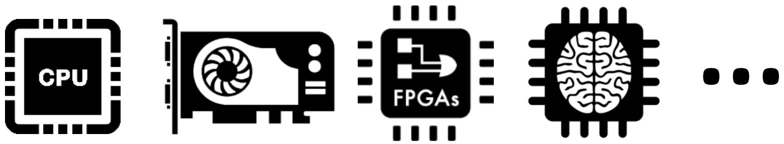
Processing in Memory (PIM) [1]

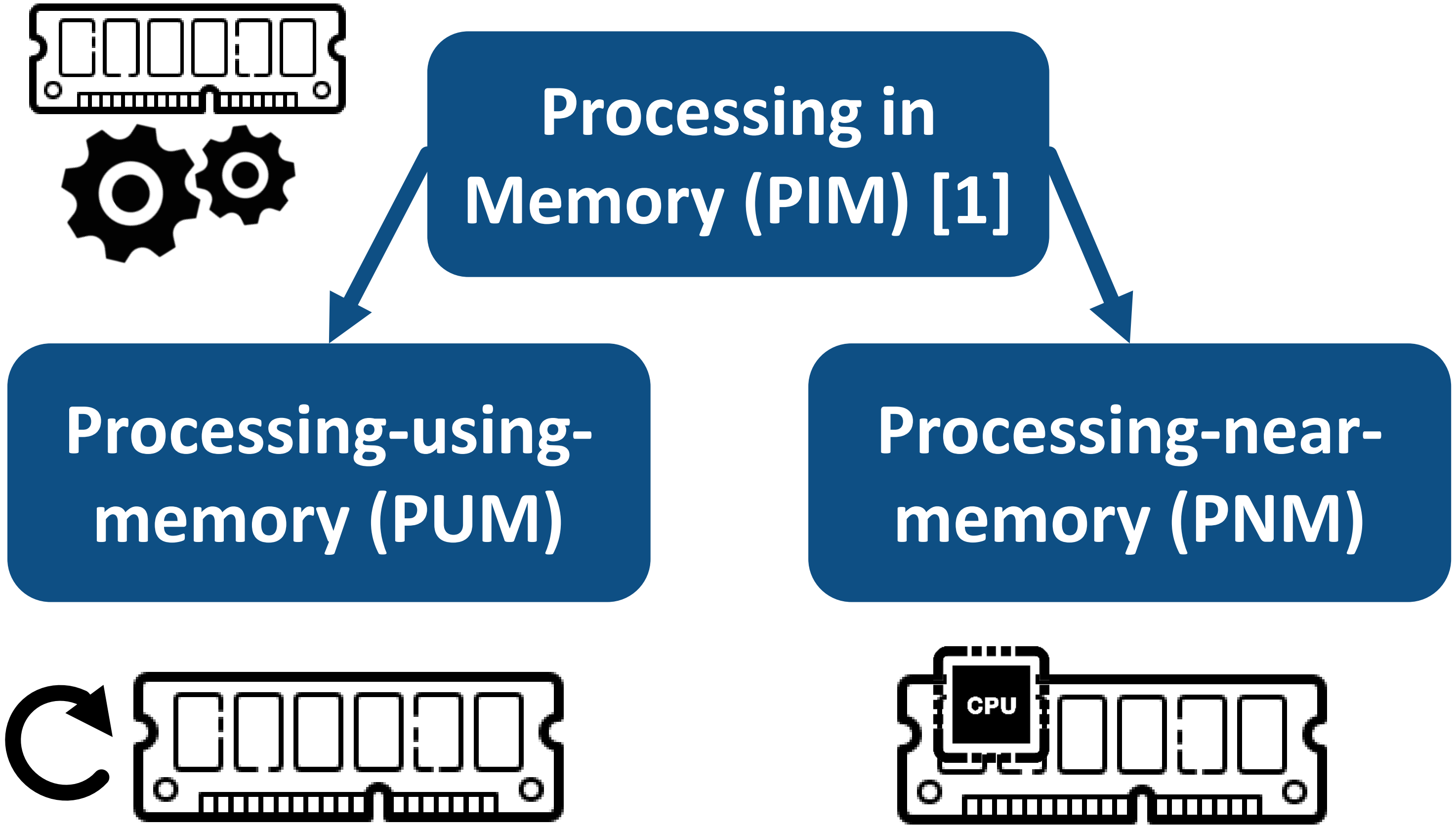
Processing-using-memory (PUM)



[1] O. Mutlu et al., A modern primer on processing in memory. 2021.

...What Hardware?





[1] O. Mutlu et al., A modern primer on processing in memory. 2021.

...What Hardware?

CPU FPGAs ...

Why PIM?

Processing in Memory (PIM) [1]

Processing-using-memory (PUM)

Processing-near-memory (PNM)

CPU

[1] O. Mutlu et al., A modern primer on processing in memory. 2021.

...What Hardware?

CPU FPGAs ...

Processing in Memory (PIM) [1]

Processing-using-memory (PUM)

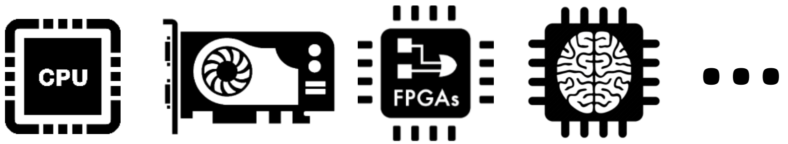
Processing-near-memory (PNM)

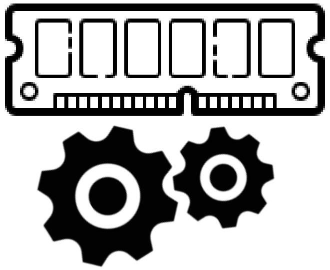
Why PIM?

A very promising tool to tackle memory bottleneck [2, ...]

[1] O. Mutlu et al., A modern primer on processing in memory. 2021.
[2] S. Ghose et al., Processing-in-Memory: A Workload-driven Perspective. IBM JRD, 2019.

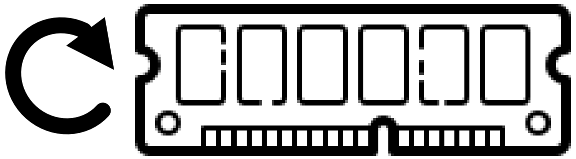
...What Hardware?



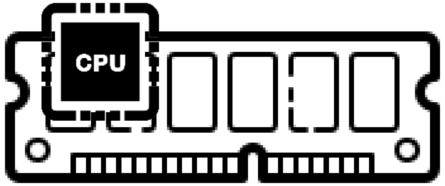


Processing in Memory (PIM) [1]

Processing-using-memory (PUM)



Processing-near-memory (PNM)



Why PIM?

A very promising tool to tackle memory bottleneck [2, ...]

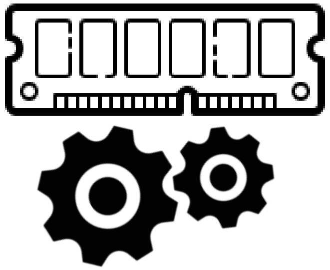
A well-understood design [2. ...]

[1] O. Mutlu et al., A modern primer on processing in memory. 2021.
[2] S. Ghose et al., Processing-in-Memory: A Workload-driven Perspective. IBM JRD, 2019.

...What Hardware?

...

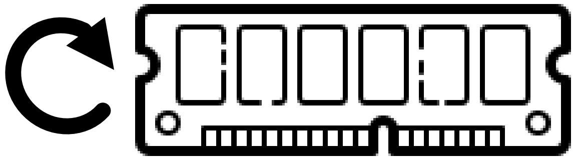
Why PIM?

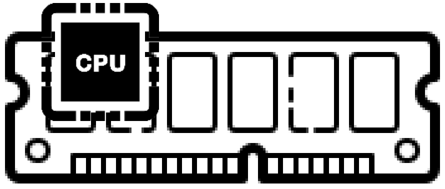


Processing in Memory (PIM) [1]

Processing-using-memory (PUM)

Processing-near-memory (PNM)





A very promising tool to tackle memory bottleneck [2, ...]

A well-understood design [2. ...]

Already used for simple graph problems with a lot of success [3, ...]

[1] O. Mutlu et al., A modern primer on processing in memory. 2021.
[2] S. Ghose et al., Processing-in-Memory: A Workload-driven Perspective. IBM JRD, 2019.
[3] C. Gui et al. A Survey on Graph Processing Accelerators: Challenges and Opportunities. JCST, 2019.

Goal: a paradigm that enables high-performance graph mining algorithms, that is…

…general (applicable to many problems)

… theoretically efficient (low work) for more performance

… easily harnesses the potential of PIM

Why PIM?

Processing in Memory (PIM) [1]

Processing-using-memory (PUM)

memory (PNM)

A very promising tool to tackle memory bottleneck [2, ...]

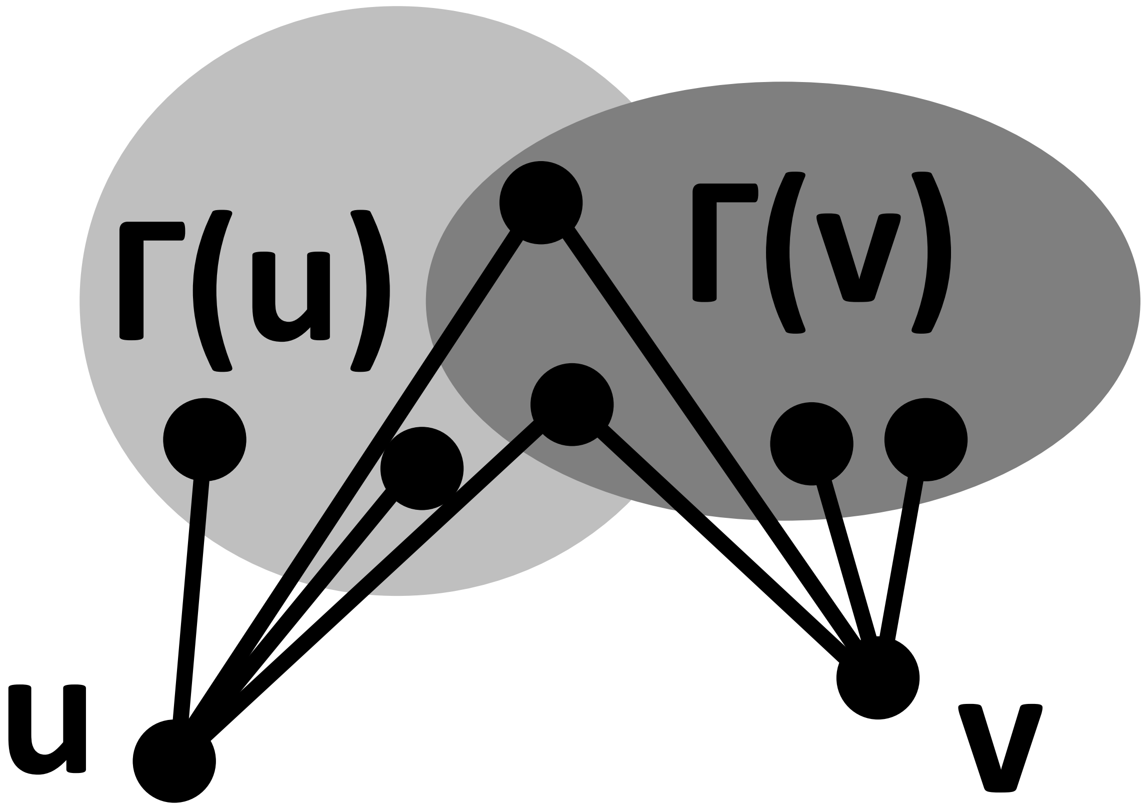
A well-understood design [2, ...]

graph problems with a lot of success [3, ...]

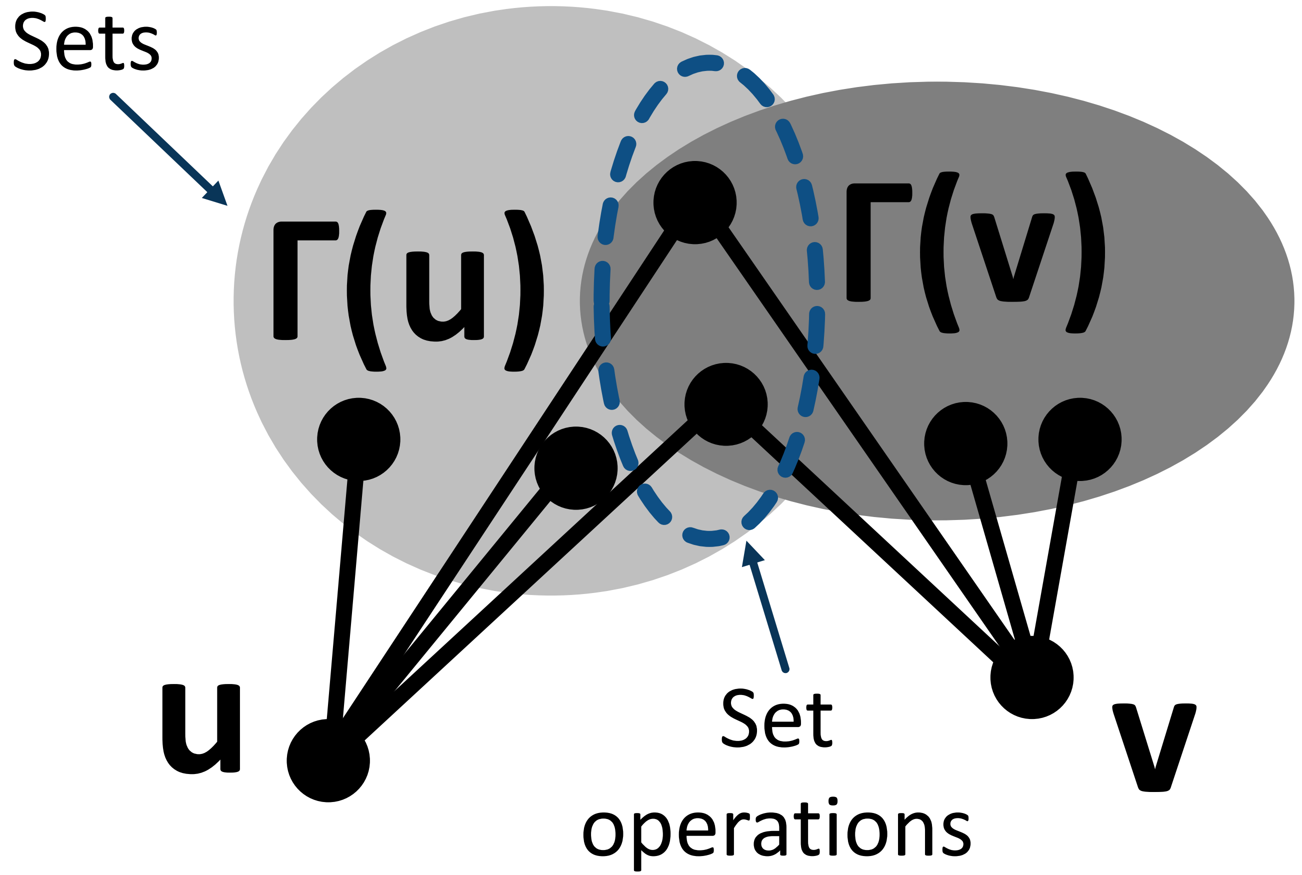
[1] O. Mutlu et al., A modern primer on processing in memory. 2021.
[2] S. Ghose et al., Processing-in-Memory: A Workload-driven Perspective. IBM JRD, 2019.
[3] C. Gui et al. A Survey on Graph Processing Accelerators: Challenges and Opportunities. JCST, 2019.

SISA: Set-Centric ISA

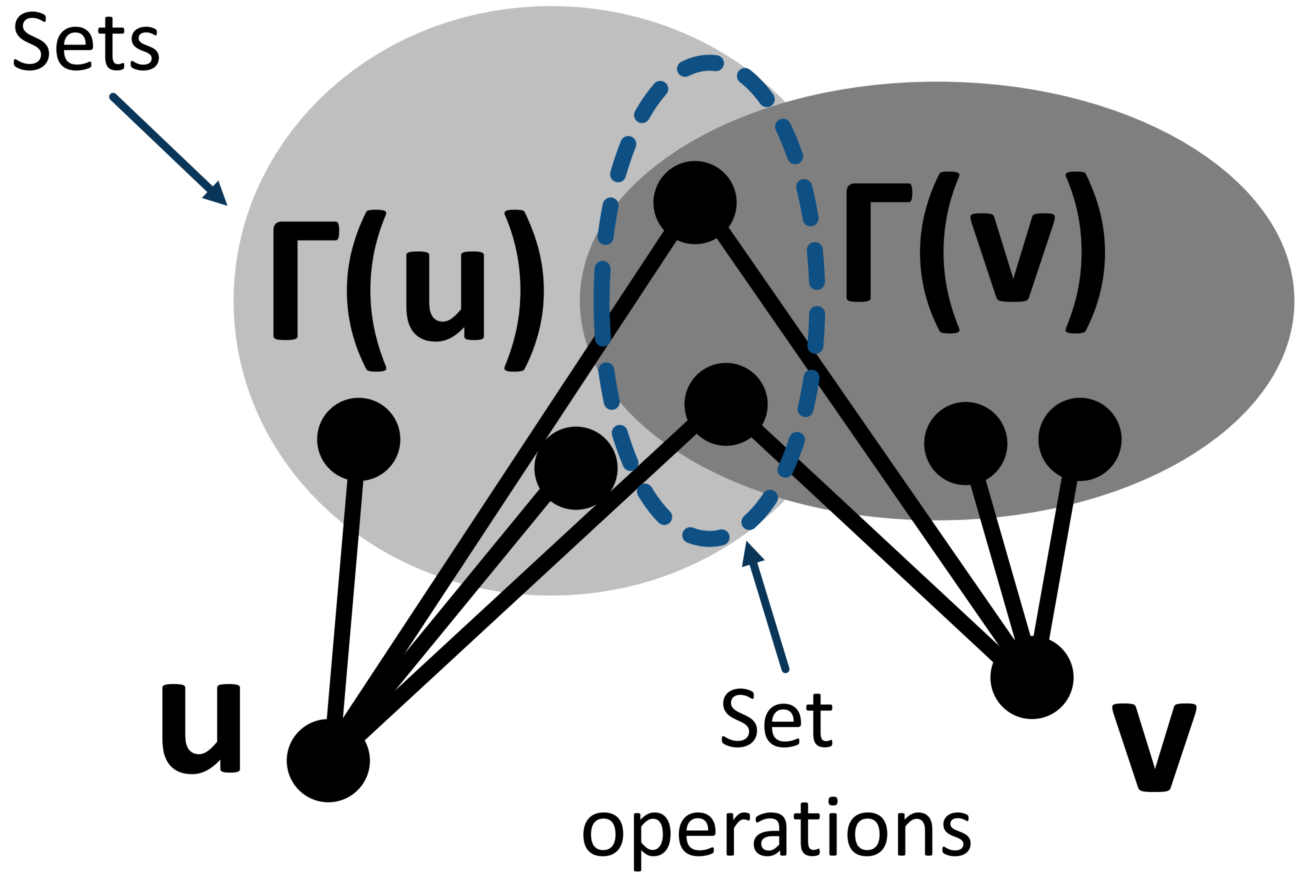
SISA: Set-Centric ISA

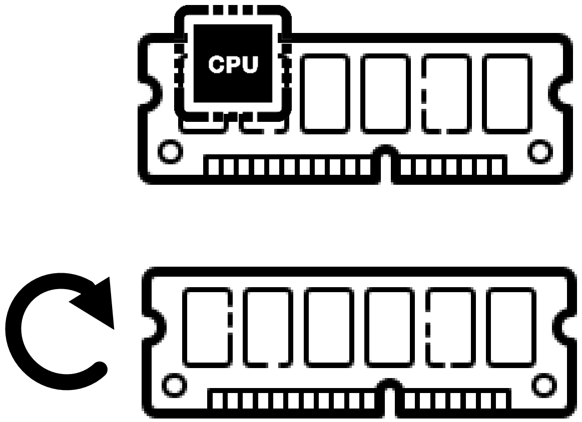


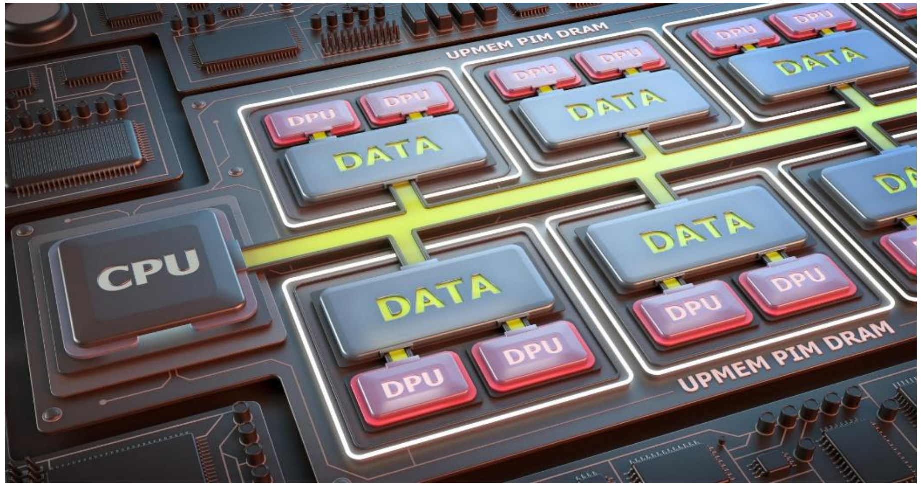
SISA: Set-Centric ISA = Set algebra



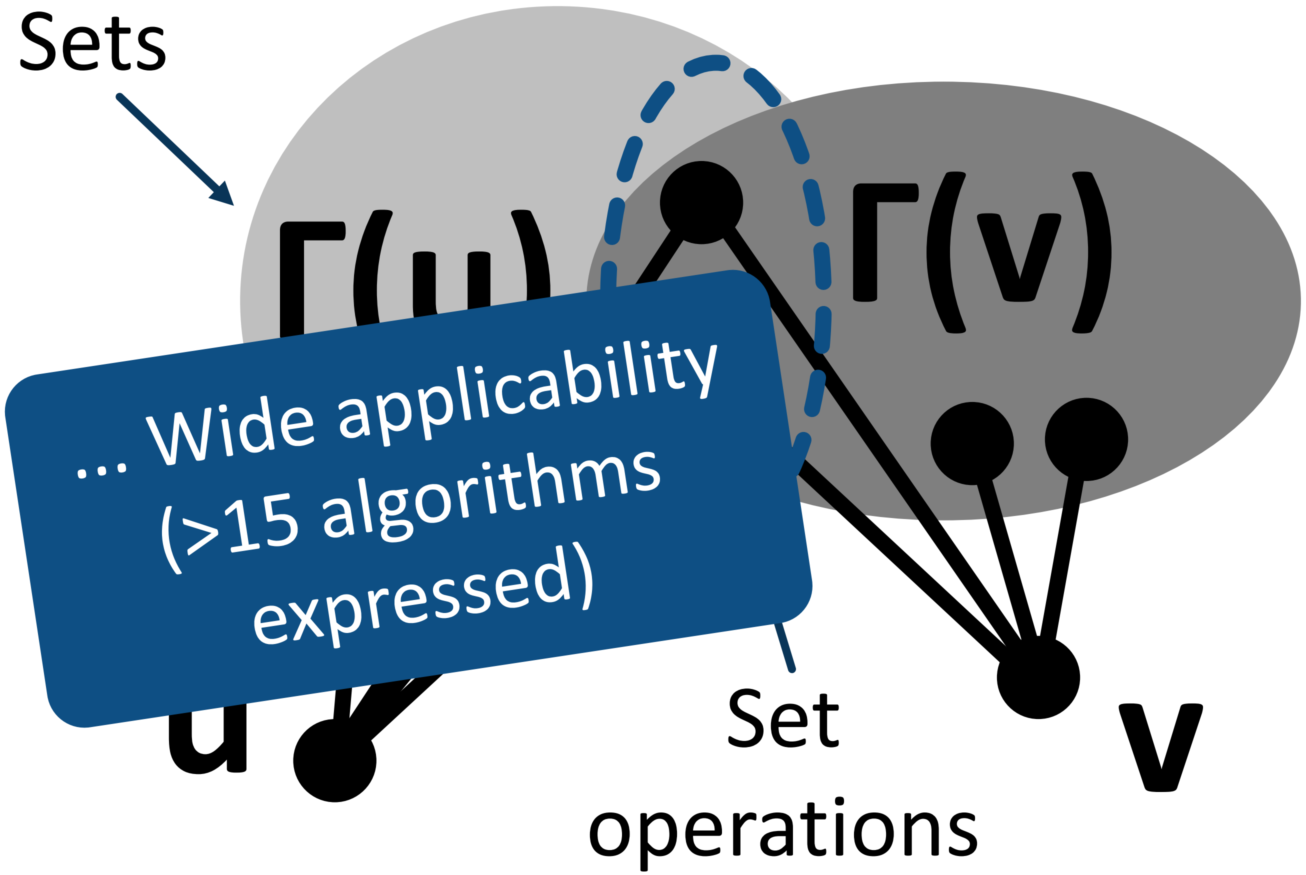
SISA: Set-Centric ISA = **Set algebra** + **Processing in Memory (PIM)**

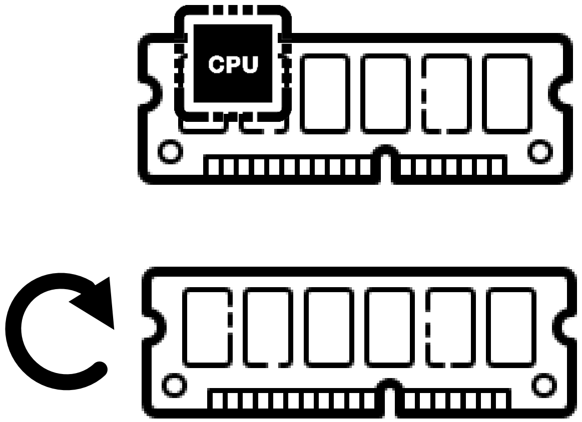


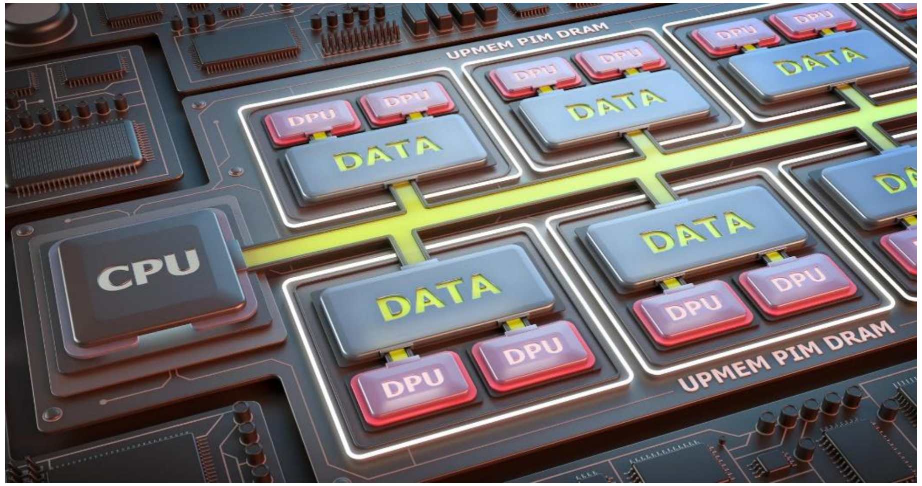




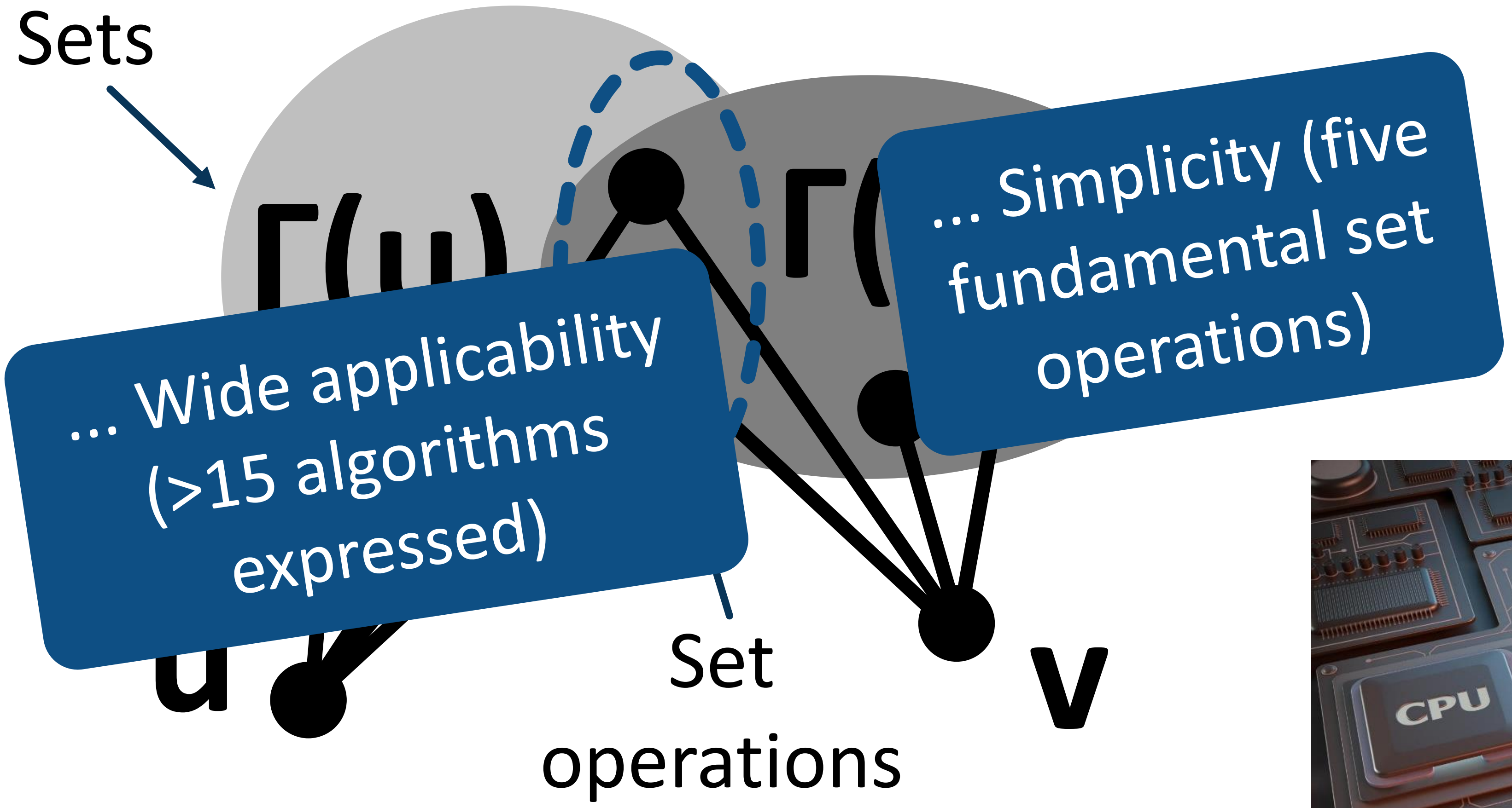
SISA: Set-Centric ISA = **Set algebra** + **Processing in Memory (PIM)**





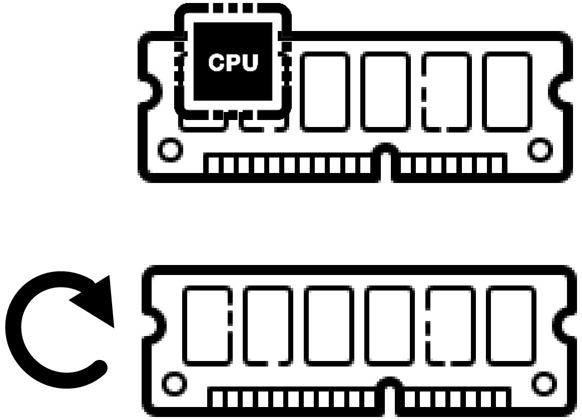


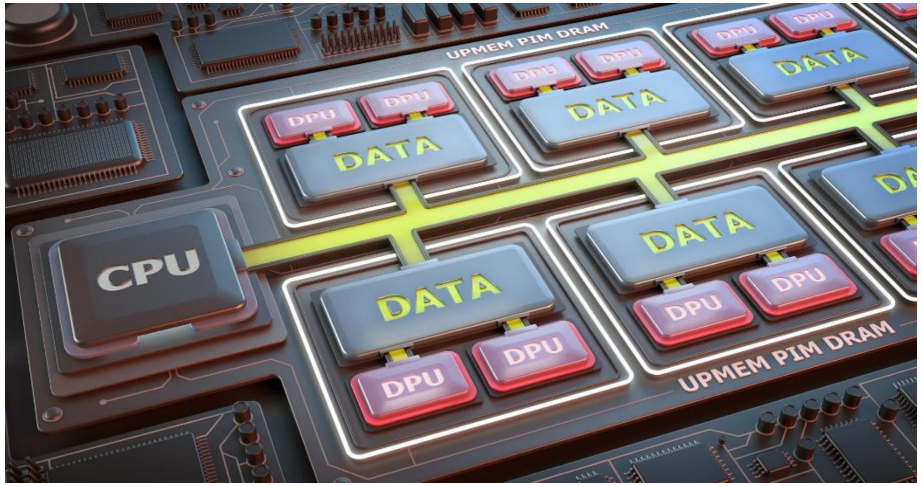
SISA: Set-Centric ISA = Set algebra + Processing in Memory (PIM)



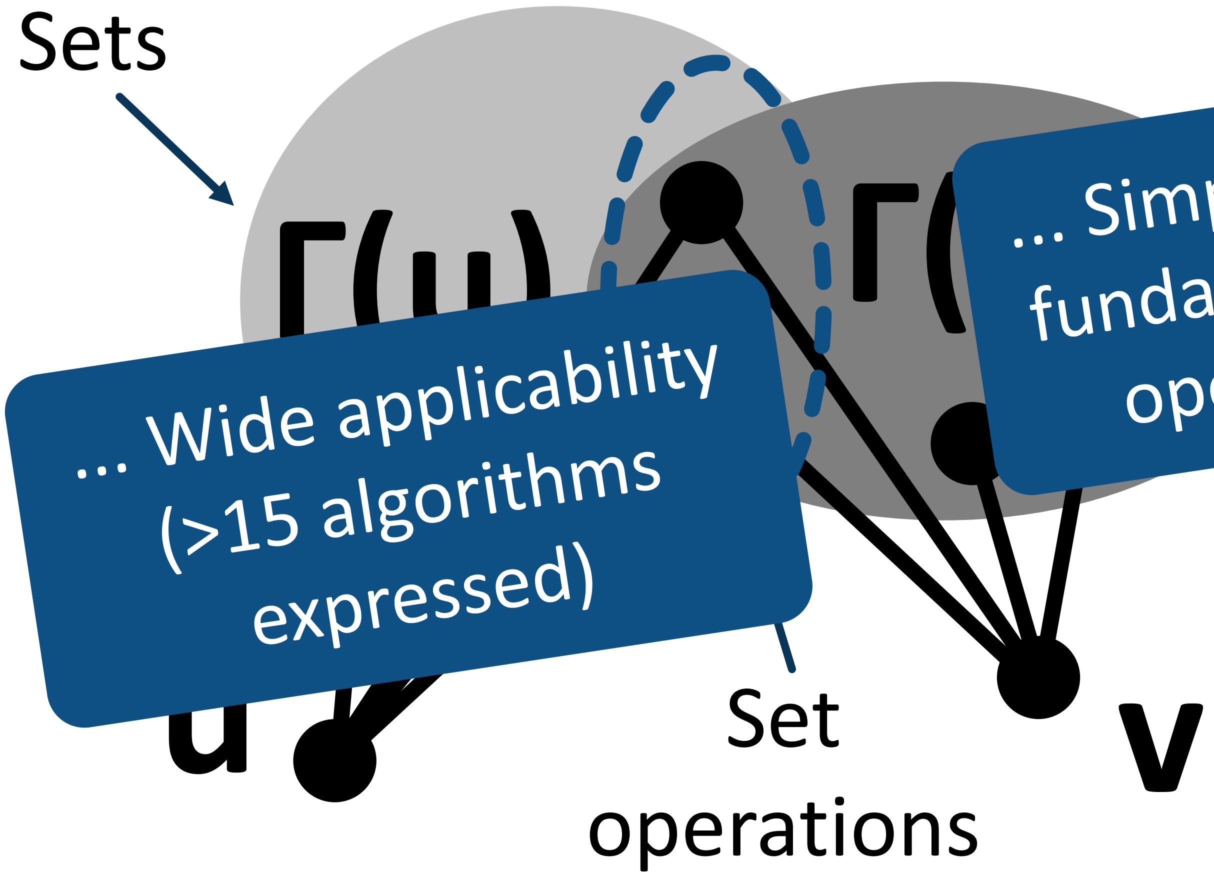
... Wide applicability (>15 algorithms expressed)

... Simplicity (five fundamental set operations)



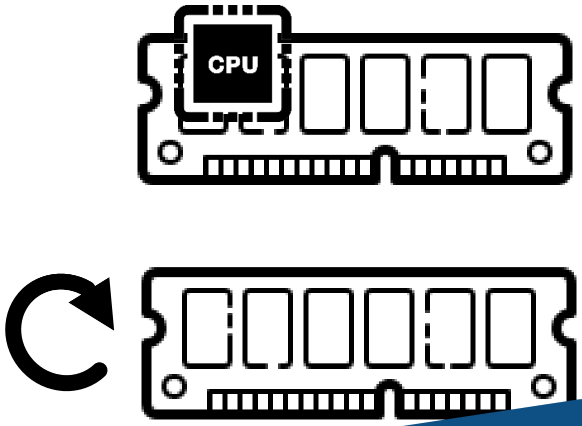


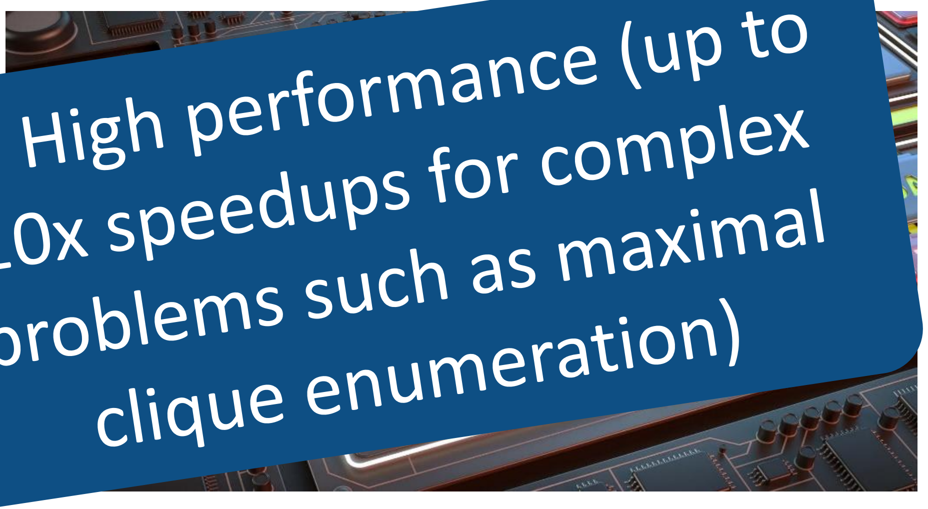
SISA: Set-Centric ISA = Set algebra + Processing in Memory (PIM)



... Wide applicability (>15 algorithms expressed)

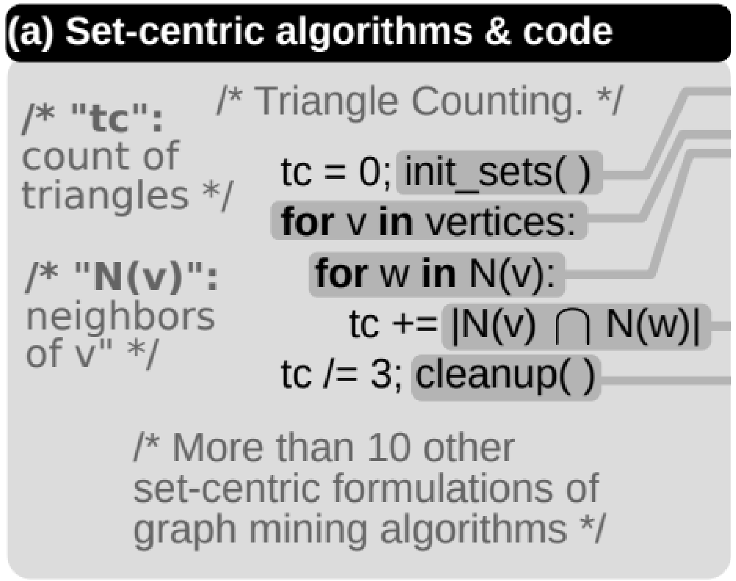
... Simplicity (five fundamental set operations)





... High performance (up to 10x speedups for complex problems such as maximal clique enumeration)

Overview


(a) Set-centric algorithms & code
/* Triangle Counting. */
/* "tc": count of triangles */
tc = 0; init_sets( )
for v in vertices:
for w in N(v):
tc += |N(v) ⋂ N(w)|
tc /= 3; cleanup( )
/* "N(v)": neighbors of v" */
/* More than 10 other set-centric formulations of graph mining algorithms */

Overview

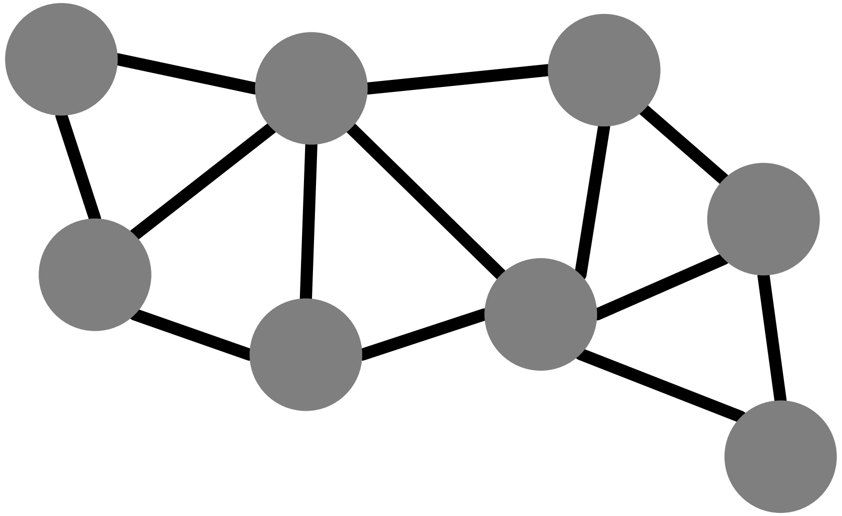
(a) Set-centric algorithms & code

/* **"tc"**: count of triangles */

/* **"N(v)"**: neighbors of v" */

```
/* Triangle Counting. */
tc = 0; init_sets( )
for v in vertices:
  for w in N(v):
    tc += |N(v) ∩ N(w)|
tc /= 3; cleanup( )
```

/* More than 10 other set-centric formulations of graph mining algorithms */



Overview

(a) Set-centric algorithms & code

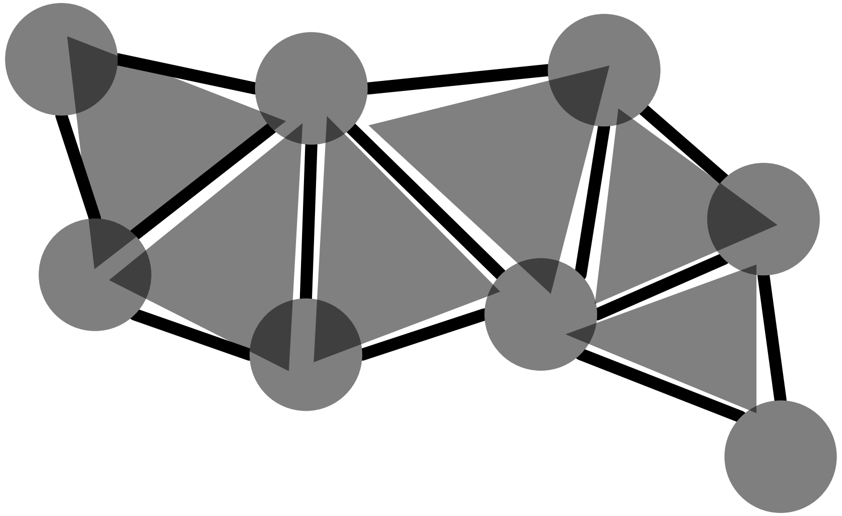
/* **"tc"**: count of triangles */

/* **"N(v)"**: neighbors of v" */

```
/* Triangle Counting. */

tc = 0; init_sets( )
for v in vertices:
  for w in N(v):
    tc += |N(v) ∩ N(w)|
tc /= 3; cleanup( )
```

/* More than 10 other set-centric formulations of graph mining algorithms */



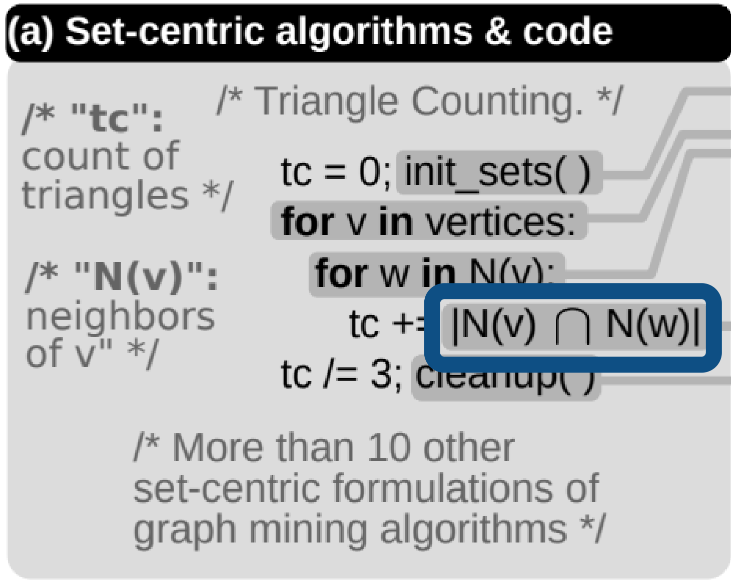
Overview

(a) Set-centric algorithms & code

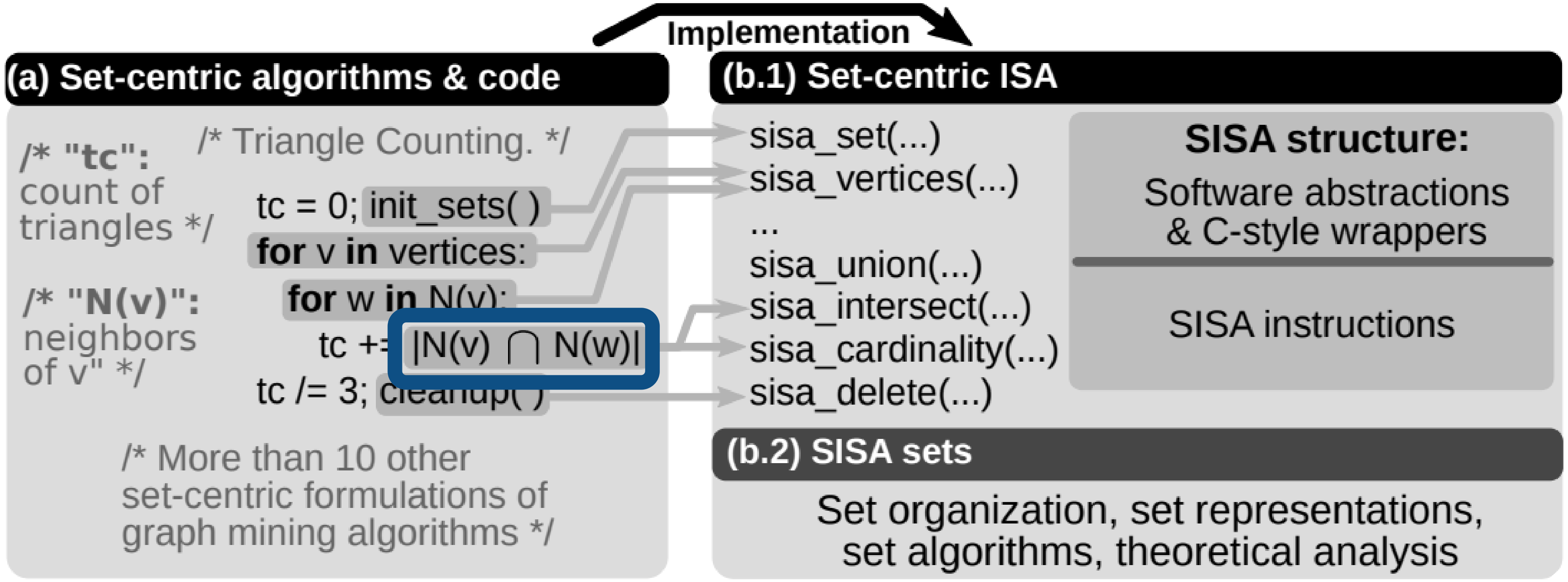
```
/* "tc":          /* Triangle Counting. */
count of
triangles */      tc = 0; init_sets( )
                  for v in vertices:
/* "N(v)":          for w in N(v):
neighbors             tc += |N(v) ∩ N(w)|
of v" */          tc /= 3; cleanup( )

    /* More than 10 other
    set-centric formulations of
    graph mining algorithms */
```

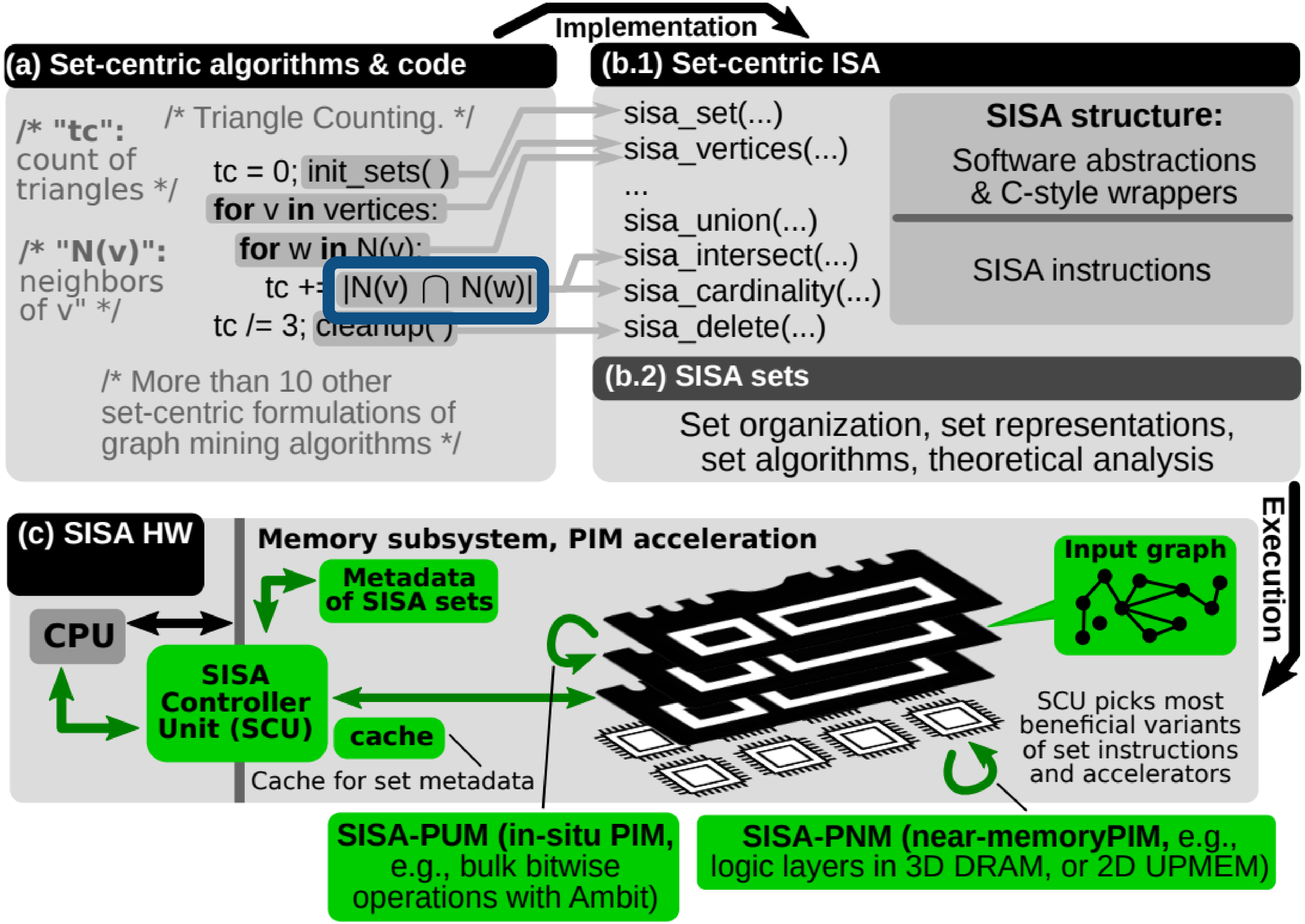
Overview


(a) Set-centric algorithms & code
/* Triangle Counting. */
/* "tc": count of triangles */
tc = 0; init_sets( )
for v in vertices:
/* "N(v)": neighbors of v" */
for w in N(v):
tc += |N(v) ⋂ N(w)|
tc /= 3; cleanup( )
/* More than 10 other set-centric formulations of graph mining algorithms */

Overview


Implementation
(a) Set-centric algorithms & code
/* Triangle Counting. */
/* "tc": count of triangles */
/* "N(v)": neighbors of v" */
tc = 0; init_sets( )
for v in vertices:
for w in N(v):
tc += |N(v) ⋂ N(w)|
tc /= 3; cleanup( )
/* More than 10 other set-centric formulations of graph mining algorithms */
(b.1) Set-centric ISA
sisa_set(...)
sisa_vertices(...)
...
sisa_union(...)
sisa_intersect(...)
sisa_cardinality(...)
sisa_delete(...)
SISA structure:
Software abstractions & C-style wrappers
SISA instructions
(b.2) SISA sets
Set organization, set representations, set algorithms, theoretical analysis

Overview


Implementation
(a) Set-centric algorithms & code
/* Triangle Counting. */
/* "tc": count of triangles */
tc = 0; init_sets( )
for v in vertices:
for w in N(v):
tc += |N(v) ∩ N(w)|
tc /= 3; cleanup( )
/* "N(v)": neighbors of v" */
/* More than 10 other set-centric formulations of graph mining algorithms */
(b.1) Set-centric ISA
sisa_set(...)
sisa_vertices(...)
...
sisa_union(...)
sisa_intersect(...)
sisa_cardinality(...)
sisa_delete(...)
SISA structure:
Software abstractions & C-style wrappers
SISA instructions
(b.2) SISA sets
Set organization, set representations, set algorithms, theoretical analysis
Execution
(c) SISA HW
Memory subsystem, PIM acceleration
CPU
Metadata of SISA sets
SISA Controller Unit (SCU)
cache
Cache for set metadata
Input graph
SCU picks most beneficial variants of set instructions and accelerators
SISA-PUM (in-situ PIM, e.g., bulk bitwise operations with Ambit)
SISA-PNM (near-memoryPIM, e.g., logic layers in 3D DRAM, or 2D UPMEM)

Our design comes with...

Our design comes with...

1

... **Set-centric paradigm & formulations** of *many* graph mining algorithms, coming with guarantees for **theoretical efficiency**

Our design comes with...

1

... **Set-centric paradigm & formulations** of *many* graph mining algorithms, coming with guarantees for **theoretical efficiency**

2

... **Set-centric ISA** with **high-performance set organization**: set representations & set algorithms

Our design comes with...

1

... **Set-centric paradigm & formulations** of *many* graph mining algorithms, coming with guarantees for **theoretical efficiency**

2

... **Set-centric ISA** with **high-performance set organization**: set representations & set algorithms

3

... **Hardware implementation of SISA** with processing-using-memory (SISA-PUM) and processing near memory (SISA-PNM)

Our design comes with...

1

... **Set-centric paradigm & formulations** of *many* graph mining algorithms, coming with guarantees for **theoretical efficiency**

2

... **Set-centric ISA** with **high-performance set organization**: set representations & set algorithms

3

... **Hardware implementation of SISA** with processing-using-memory (SISA-PUM) and processing near memory (SISA-PNM)

Set-Centric Paradigm & Formulations

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Set-Centric Paradigm & Formulations

Essence: decompose a graph mining algorithm into set algebra building blocks, and map them to SISA instructions for PIM acceleration

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Set-Centric Paradigm & Formulations

Essence: decompose a graph mining algorithm into set algebra building blocks, and map them to SISA instructions for PIM acceleration

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Set-Centric Paradigm & Formulations

Essence: decompose a graph mining algorithm into set algebra building blocks, and map them to SISA instructions for PIM acceleration

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Prevalence of set operations in graph mining algorithms & problems: we develop 15+ set-centric formulations

Set-Centric Paradigm & Formulations

Essence: decompose a graph mining algorithm into set algebra building blocks, and map them to SISA instructions for PIM acceleration

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Wide applicability

Prevalence of set operations in graph mining algorithms & problems: we develop 15+ set-centric formulations

Set-Centric Paradigm & Formulations

Essence: decompose a graph mining algorithm into set algebra building blocks, and map them to SISA instructions for PIM acceleration

Prevalence of set operations in graph mining algorithms & problems: we develop 15+ set-centric formulations

Simplicity

Wide applicability

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Set-Centric Paradigm & Formulations

Essence: decompose a graph mining algorithm into set algebra building blocks, and map them to SISA instructions for PIM acceleration

Simplicity

Wide applicability

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Prevalence of set operations in graph mining algorithms & problems: we develop 15+ set-centric formulations

There are only a few set operations, they are easy to reason about

Set-Centric Paradigm & Formulations

Essence: decompose a graph mining algorithm into set algebra building blocks, and map them to SISA instructions for PIM acceleration

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Simplicity

Wide applicability

Prevalence of set operations in graph mining algorithms & problems: we develop 15+ set-centric formulations

We can use very efficient (e.g., work optimal) graph mining algorithms and simply expose set operations

There are only a few set operations, they are easy to reason about

Set-Centric Paradigm & Formulations

Essence: decompose a graph mining algorithm into set algebra building blocks, and map them to SISA instructions for PIM acceleration

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Simplicity

Efficiency

Wide applicability

Prevalence of set operations in graph mining algorithms & problems: we develop 15+ set-centric formulations

We can use very efficient (e.g., work optimal) graph mining algorithms and simply expose set operations

There are only a few set operations, they are easy to reason about

Set-Centric Paradigm & Formulations

Essence: decompose a graph mining algorithm into set algebra building blocks, and map them to SISA instructions for PIM acceleration

Prevalence of set operations in graph mining algorithms & problems: we develop 15+ set-centric formulations

Simplicity

Wide applicability

Efficiency

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

We can use very efficient (e.g., work optimal) graph mining algorithms and simply expose set operations

There are only a few set operations, they are easy to reason about

One can select most efficient variant of set algorithm (for a given scenario)

Set-Centric Paradigm & Formulations

Essence: decompose a graph mining algorithm into set algebra building blocks, and map them to SISA instructions for PIM acceleration

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Simplicity

High performance

Efficiency

Wide applicability

Prevalence of set operations in graph mining algorithms & problems: we develop 15+ set-centric formulations

We can use very efficient (e.g., work optimal) graph mining algorithms and simply expose set operations

There are only a few set operations, they are easy to reason about

One can select most efficient variant of set algorithm (for a given scenario)

Analysis of Expressiveness & Comparison to Other Paradigms

Abstraction or programming model	Example design	Underlying algebra?	Key element	Key operations	Pattern M.				Learning				“Low-c.”				Remarks
					mc	kc	ds	si	vs	lp	cl	av	tc	bf	cc	pr	
Vertex-centric (ver-c)																	
Edge-centric (edge-c)																	
Array maps																	
GraphBLAS																	
GNN (graph neural networks)																	
Pattern matching																	
Joins																	
Set-Centric [This work]																	

Analysis of Expressiveness & Comparison to Other Paradigms

Abstraction or programming model	Example design	Underlying algebra?	Key element	Key operations	Pattern M.				Learning				"Low-c."				Remarks
					mc	kc	ds	si	vs	lp	cl	av	tc	bf	cc	pr	
Vertex-centric (ver-c)																	
Edge-centric (edge-c)																	
Array maps																	
GraphBLAS																	
GNN (graph neural networks)																	
Pattern matching																	
Joins																	
Set-Centric [This work]																	

Analysis of Expressiveness & Comparison to Other Paradigms

Clique/subgraph enumeration, isomorphism, ...

Abstraction or programming model	Example design	Underlying algebra?	Key element	Key operations	Pattern M.				Learning			"Low-c."					Remarks
					mc	kc	ds	si	vs	lp	cl	av	tc	bf	cc	pr	
Vertex-centric (ver-c)																	
Edge-centric (edge-c)																	
Array maps																	
GraphBLAS																	
GNN (graph neural networks)																	
Pattern matching																	
Joins																	
Set-Centric [This work]																	

Analysis of Expressiveness & Comparison to Other Paradigms

Clique/subgraph enumeration, isomorphism, ...

Clustering, link prediction, ...

Abstraction or programming model	Example design	Underlying algebra?	Key element	Key operations	Pattern M.				Learning			"Low-c."					Remarks
					mc	kc	ds	si	vs	lp	cl	av	tc	bf	cc	pr	
Vertex-centric (ver-c)																	
Edge-centric (edge-c)																	
Array maps																	
GraphBLAS																	
GNN (graph neural networks)																	
Pattern matching																	
Joins																	
Set-Centric [This work]																	

Analysis of Expressiveness & Comparison to Other Paradigms

Clique/subgraph enumeration, isomorphism, ...

Clustering, link prediction, ...

BFS, PageRank, ...

Abstraction or programming model	Example design	Underlying algebra?	Key element	Key operations	Pattern M.				Learning			"Low-c."				Remarks
					mc	kc	ds	si	vs	lp	cl	av	tc	bf	cc	pr	
Vertex-centric (ver-c)																
Edge-centric (edge-c)																
Array maps																
GraphBLAS																
GNN (graph neural networks)																
Pattern matching																
Joins																
Set-Centric [This work]																

Analysis of Expressiveness & Comparison to Other Paradigms

Clique/subgraph enumeration, isomorphism, ...

Clustering, link prediction, ...

BFS, PageRank, ...

Abstraction or programming model	Example design	Underlying algebra?	Key element	Key operations	Pattern M.				Learning				"Low-c."				Remarks
					mc	kc	ds	si	vs	lp	cl	av	tc	bf	cc	pr	
Vertex-centric (ver-c)					✗	✗	✗	✗	✗	✗	✗	✗	partial*	full	partial†	full	
Edge-centric (edge-c)					✗	✗	✗	✗	✗	✗	✗	✗	partial*	partial*	partial*	full	
Array maps					✗	✗	full	✗	✗	✗	partial*	✗	partial	full	full	full	
GraphBLAS					✗	✗	✗	partial*	✗	✗	✗	✗	full	full	partial†	full	
GNN (graph neural networks)					✗	✗	✗	partial†	partial	full	full	full	✗	✗	✗	partial	
Pattern matching					partial*	partial*	partial*	partial*	✗	✗	✗	✗	partial*	✗	✗	✗	
Joins					✗	partial*	partial*	✗	partial*	partial*	partial*	✗	partial*	✗	partial*	partial*	
Set-Centric [This work]					full	full	full	full	full	full	full	full	full	✗	✗	✗	

Analysis of Expressiveness & Comparison to Other Paradigms

Clique/subgraph enumeration, isomorphism, ...

Clustering, link prediction, ...

BFS, PageRank, ...

Abstraction or programming model	Example design	Underlying algebra?	Key element	Key operations	Pattern M.				Learning				"Low-c."				Remarks
					mc	kc	ds	si	vs	lp	cl	av	tc	bf	cc	pr	
Vertex-centric (ver-c)					✘	✘	✘	✘	✘	✘	✘	✘	▄*	▮	▄†	▮	
Edge-centric (edge-c)					✘	✘	✘	✘	✘	✘	✘	✘	▄*	▄*	▄*	▮	
Array maps					✘	✘	▮	✘	✘	✘	▄*	✘	▄	▮	▮	▮	
GraphBLAS					✘	✘	✘	▄*	✘	✘	✘	✘	▮	▮	▄†	▮	
GNN (graph neural networks)					✘	✘	✘	▄†	▄	▮	▮	▮	✘	✘	✘	▄	
Pattern matching					▄*	▄*	▄*	▄*	✘	✘	✘	✘	▄*	✘	✘	✘	
Joins					✘	▄*	▄*	✘	▄*	▄*	▄*	✘	▄*	✘	▄*	▄*	
Set-Centric [This work]	**SISA [This work]**	▮ (set)	Sets of vertices/edges	Set operations	▮	▮	▮	▮	▮	▮	▮	▮	▮	✘	✘	✘	

Analysis of Expressiveness & Comparison to Other Paradigms

Clique/subgraph enumeration, isomorphism, ...

Clustering, link prediction, ...

BFS, PageRank, ...

Abstraction or programming model	Example design	Underlying algebra?	Key element	Key operations	Pattern M.				Learning				"Low-c."				Remarks
					mc	kc	ds	si	vs	lp	cl	av	tc	bf	cc	pr	
Vertex-centric (ver-c)	PowerGraph	✗	Vertex + its neighbors	Vertex kernel	✗	✗	✗	✗	✗	✗	✗	✗	🪫*	🔋	🪫†	🔋	
Edge-centric (edge-c)					✗	✗	✗	✗	✗	✗	✗	✗	🪫*	🪫*	🪫*	🔋	
Array maps					✗	✗	🔋	✗	✗	✗	🪫*	✗	🪫	🔋	🔋	🔋	
GraphBLAS					✗	✗	✗	🪫*	✗	✗	✗	✗	🔋	🔋	🪫†	🔋	
GNN (graph neural networks)					✗	✗	✗	🪫†	🪫	🔋	🔋	🔋	✗	✗	✗	🪫	
Pattern matching					🪫*	🪫*	🪫*	🪫*	✗	✗	✗	✗	🪫*	✗	✗	✗	
Joins					✗	🪫*	🪫*	✗	🪫*	🪫*	🪫*	✗	🪫*	✗	🪫*	🪫*	
Set-Centric [This work]	**SISA [This work]**	🔋 (set)	Sets of vertices/edges	Set operations	🔋	🔋	🔋	🔋	🔋	🔋	🔋	🔋	🔋	✗	✗	✗	

Analysis of Expressiveness & Comparison to Other Paradigms

Clique/subgraph enumeration, isomorphism, ...

Clustering, link prediction, ...

BFS, PageRank, ...

Abstraction or programming model	Example design	Underlying algebra?	Key element	Key operations	Pattern M.				Learning				"Low-c."				Remarks
					mc	kc	ds	si	vs	lp	cl	av	tc	bf	cc	pr	
Vertex-centric (ver-c)	PowerGraph	✗	Vertex + its neighbors	Vertex kernel	✗	✗	✗	✗	✗	✗	✗	✗	partial*	full	partial†	full	
Edge-centric (edge-c)					✗	✗	✗	✗	✗	✗	✗	✗	partial*	partial*	partial*	full	
Array maps					✗	✗	full	✗	✗	✗	partial*	✗	partial	full	full	full	
GraphBLAS	GraphMat	full (linear)	Matrix, vector	SpMV, SpMSpM	✗	✗	✗	partial*	✗	✗	✗	✗	full	full	partial†	full	
GNN (graph neural networks)					✗	✗	✗	partial†	partial	full	full	full	✗	✗	✗	partial	
Pattern matching					partial*	partial*	partial*	partial*	✗	✗	✗	✗	partial*	✗	✗	✗	
Joins					✗	partial*	partial*	✗	partial*	partial*	partial*	✗	partial*	✗	partial*	partial*	
Set-Centric [This work]	**SISA [This work]**	full (set)	Sets of vertices/edges	Set operations	full	full	full	full	full	full	full	full	full	✗	✗	✗	

Analysis of Expressiveness & Comparison to Other Paradigms

Analysis of Expressiveness & Comparison to Other Paradigms

```
/* Input: A graph G = (V,E). Output: Clustering C ⊆ E */
for e = (v,u) ∈ E [in par] do: //τ is a user-defined threshold
  if |N(v) ∩ N(u)| > τ: C = C ∪ {e
```

Algorithm 11: Jarvis-Patrick

```
/* Input: A graph G. Output: Degenera
i = 0
```

```
/* Input: target graph (G), minimum support / count of a found pattern (σ).
 * Output: sets of frequent subgraphs of sizes 1,2,...,k (F1,F2,...,Fk).*/
F1 = V; k = 2 //k = 2 means we start recursion from edges.
//Use all subgraphs in Fk−1 to generate candidates of size k:
while Fk−1 ≠ ∅ do: //Ck (below) are candidate subgraphs of size k
  Fk = ∅; Ck = candidate_gen(Fk−1) //Use any selected kernel[128]
  foreach g ∈ Ck do:
    cnt = SI(g, G) //For set operations in SI, see Algorithm 7
    if cnt ≥ σn and g ∉ Fk: Fk ∪= g
  k++
```

Algorithm 8: Frequent subgraph mining [128].

```
/* Input: A graph G = (V,E). Output: Effectiveness eff
 * of a given prediction scheme. */
Erndm = /* Random subset of E */
Esparse = E \ Erndm /* Edges in E after removing Erndm */
//For each e ∈ (V × V) \ Esparse, derive score S(e) that
//determines the chance that e appears in future. Here,
//one can use any vertex similarity scheme S from § 5.2.1.
for e = (v,u) ∈ (V × V) \ Esparse [in par] do: compute S(v,u)
Epredict = /* Pick selected top edges with the highest S scores.
  */
eff = |Epredict ∩ Erndm| //Derive the effectiveness.
```

Algorithm 10: Link prediction testing.

```
 Output: All k-clique-stars, S.*/
d k-cliques (e.g., with Table 4)*/
t with identified k-clique-stars.
```

```
    for v ∈ P \ N(u) do:
      BKPivot( R ∪ {v}, P ∩ N(v), X ∩ N(
```

```
/* Input: A graph G. Output: S contains the maximal k-c
C = /* First, find (k+1)-cliques (use Listing 3
S = /* Empty map where the keys are k-cliques a
      k-clique-stars */
for c ∈ C [in par] do: //For each (k+1)-clique c
  for v ∈ c do: //for each vertex in clique c...
    S[c \ {v}] ∪= c //Add c to a k-clique-star.
```

Algorithm 5: k-clique-star listing (our

```
attern graph: output M(s); return;
date pairs to be added to M(s) */
    checkCore = ... original ... core rule */
    checkTerm = |N1(v1) ∩ T1(s)| ≥ |N2(v2) ∩ T2(s)|
    checkNew = |N1(v1) \ (M1(s) ∪ T1(s))| ≥ |N2(v2) \ (M2(s) ∪ T2(s)
    checkFeasibility = checkCore ∧ checkTerm ∧ checkNew
    checkSemantic = verify_labels(v1, v2, s) //If we use labels
    checkFeasibility = checkFeasibility ∧ checkSemantic //If we use
      labels.
    if checkFeasibility : s′ = NewState(s, v1, v2); Match(s′)
//Check if labeling of v1 and v2 and their neighborhoods match
bool verify_labels(v1, v2, s):
  forall v′1 ∈ N1(v1) ∩ M1(s): forall (v′1, v′2) ∈ M(s):
    if (L(v1) != L(v2)) or (L(v1,v′1) != L(v2,v′2)): return false
  return true
```

Algorithm 7: Subgraph isomorphism [69].

```
/* Input: A graph G. Output: Similarity S ∈ ℝ of sets A,B.
 * Most often, A and B are neighborhoods N(u) and N(v)
 * of vertices u and v. */
//Jaccard similarity:
S(A,B) = |A ∩ B| / |A ∪ B| = |A ∩ B| / (|A| + |B| - |A ∩ B|)
//Overlap similarity:
S(A,B) = |A ∩ B| / min(|A|, |B|)
//Certain measures are only defined for neighborhoods:
S(v,u) = Σw(1/log|N(w)|) //where w ∈ N(v) ∩ N(u); Adamic Adar
S(v,u) = Σw(1/|N(w)|) //where w ∈ N(v) ∩ N(u); Resource Alloc.
S(v,u) = |N(v) ∩ N(u)| //Common Neighbors
S(v,u) = |N(v) ∪ N(u)| //Total Neighbors
```

Algorithm 9: Example vertex similarity measures [148].

Our design comes with...

1 ... **Set-centric paradigm & formulations** of *many* graph mining algorithms, coming with guarantees for **theoretical efficiency**

2 ... **Set-centric ISA** with **high-performance set organization**: set representations & set algorithms

3 ... **Hardware implementation of SISA** with processing-using-memory (SISA-PUM) and processing near memory (SISA-PNM)

Our design comes with...

1

... **Set-centric paradigm & formulations** of *many* graph mining algorithms, coming with guarantees for **theoretical efficiency**

2

... **Set-centric ISA** with **high-performance set organization**: set representations & set algorithms

3

... **Hardware implementation of SISA** with processing-using-memory (SISA-PUM) and processing near memory (SISA-PNM)

Our design comes with...

1

... **Set-centric paradigm & formulations** of *many* graph mining algorithms, coming with guarantees for **theoretical efficiency**

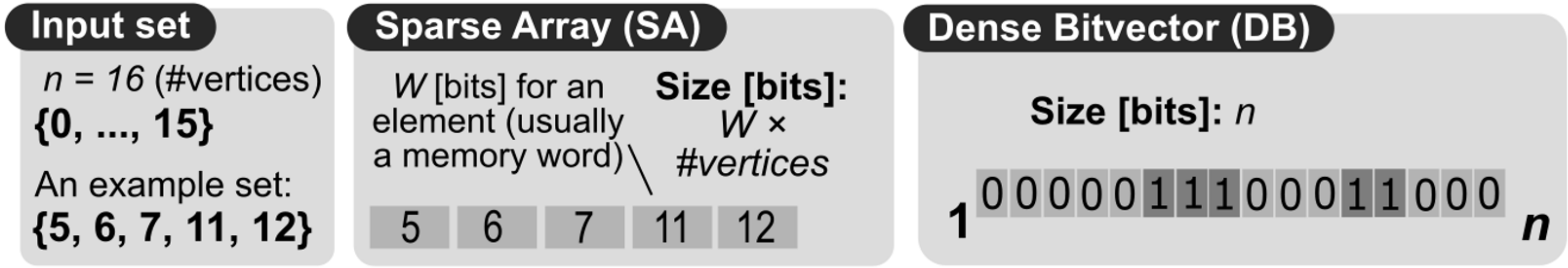
2

... **Set-centric ISA** with **high-performance set organization**: set representations & set algorithms

3

... **Hardware implementation of SISA** with processing-using-memory (SISA-PUM) and processing near memory (SISA-PNM)

Set Representations



Set Representations

Input set

$n = 16$ (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: $W \times$ *#vertices*

5	6	7	11	12

Dense Bitvector (DB)

Size [bits]: n

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 n

Set Algorithms

SA, SA (similar sizes)

SA, SA (sizes vary a lot)

SA, DB

DB, DB

Other set operations have similar variants

Set Representations

Input set

n = *16* (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

5 6 7 11 12

Dense Bitvector (DB)

Size [bits]: *n*

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 *n*

Set Algorithms

SA, SA (similar sizes)

SA, SA (sizes vary a lot)

SA, DB

DB, DB

Other set operations have similar variants

Set Representations

Input set

n = *16* (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

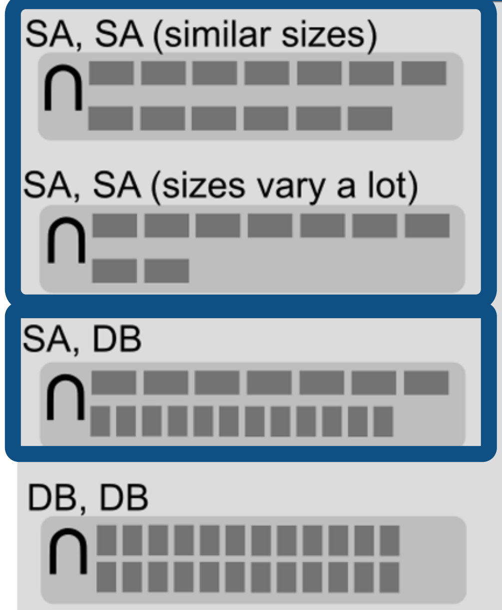
5	6	7	11	12

Dense Bitvector (DB)

Size [bits]: *n*

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 *n*

Set Algorithms



Other set operations have similar variants

Set Representations

Input set

n = 16 (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

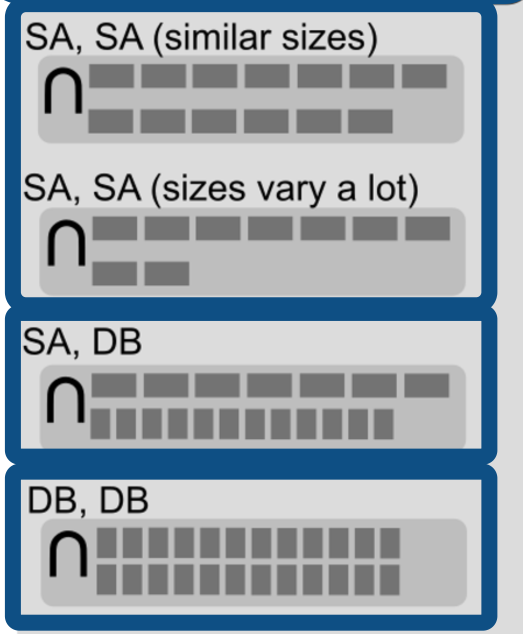
5	6	7	11	12

Dense Bitvector (DB)

Size [bits]: *n*

1 | 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 | *n*

Set Algorithms



Other set operations have similar variants

Set Representations

Input set

n = 16 (#vertices)
{0, ..., 15}
An example set:
{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

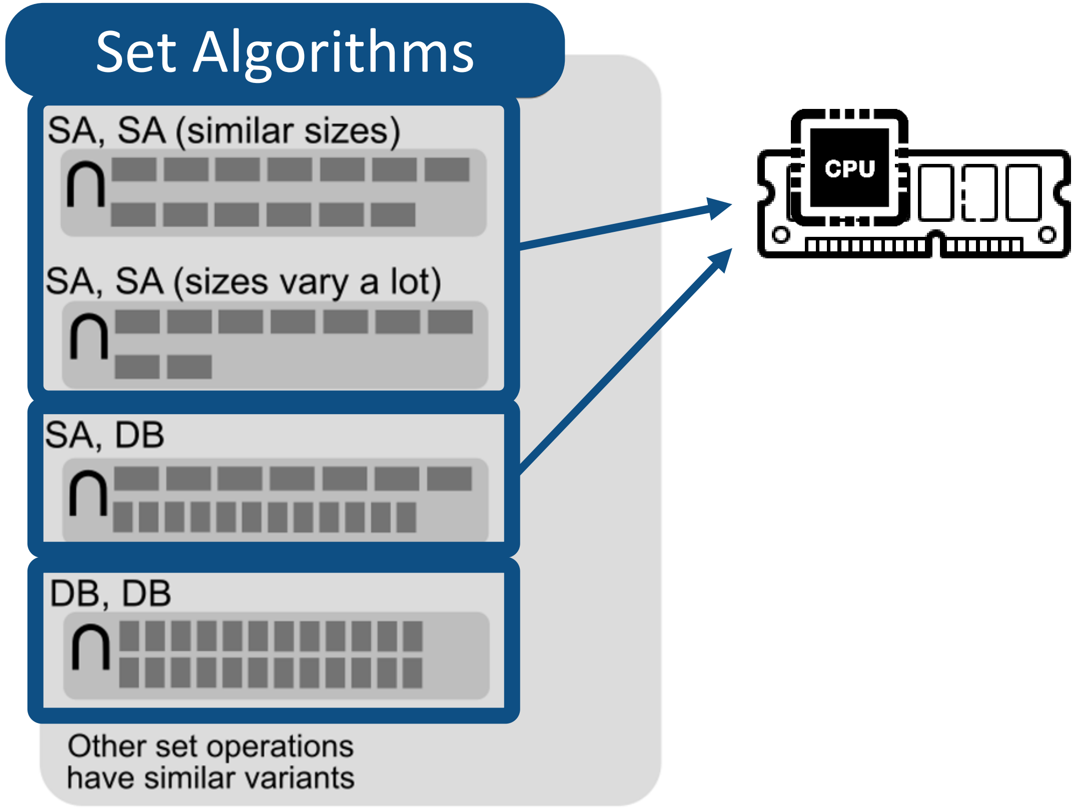
Size [bits]: *W* × *#vertices*

5	6	7	11	12

Dense Bitvector (DB)

Size [bits]: *n*

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 *n*



Set Representations

Input set

n = 16 (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

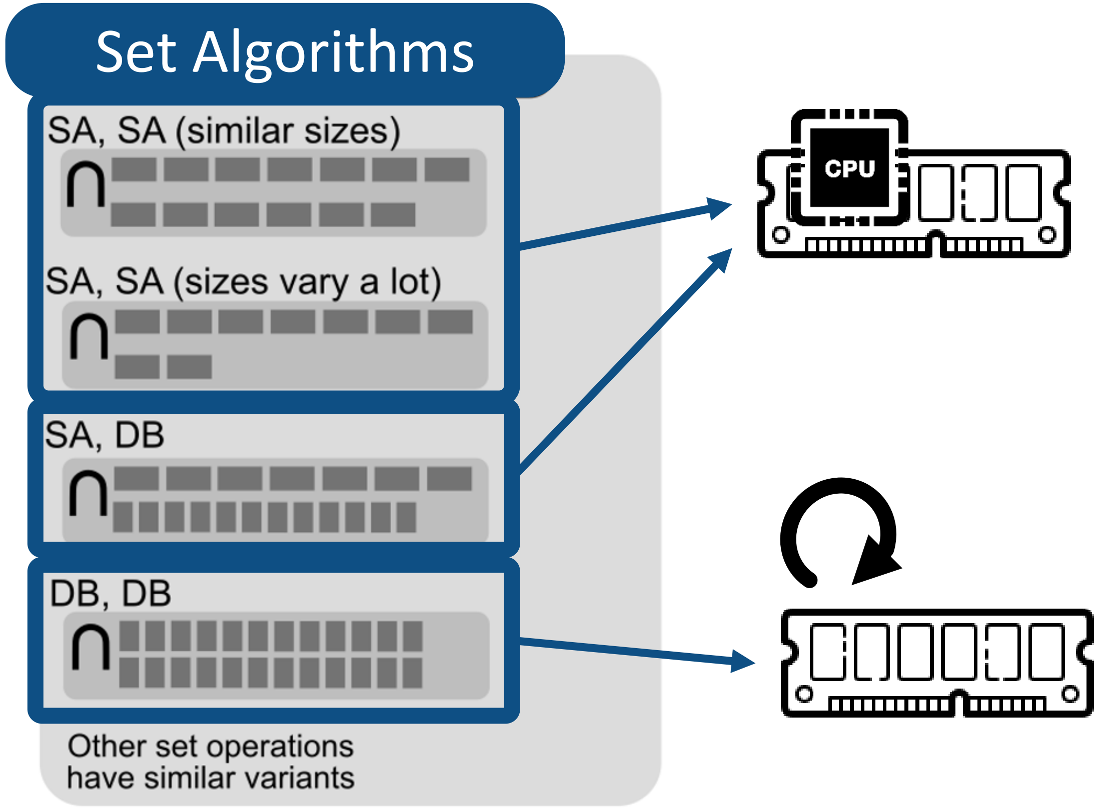
Size [bits]: *W* × *#vertices*

5	6	7	11	12

Dense Bitvector (DB)

Size [bits]: *n*

1 | 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 | ***n***



Set Representations

Input set

n = 16 (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

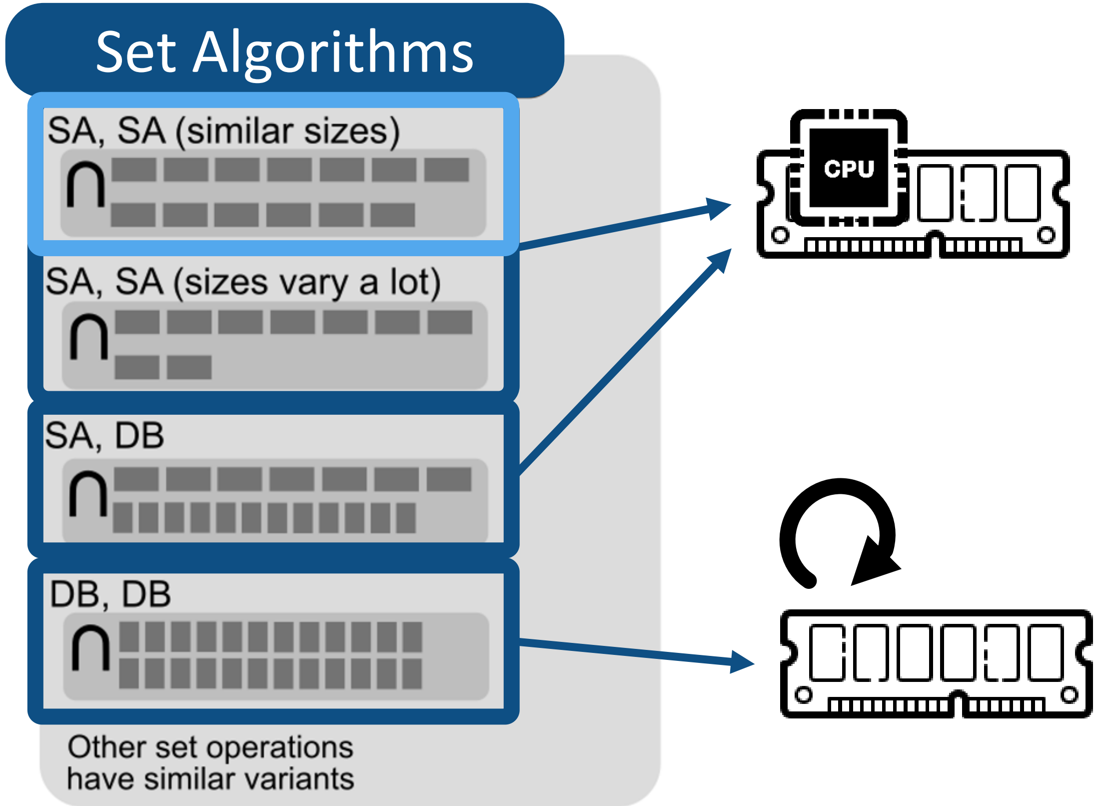
Size [bits]: *W* × *#vertices*

5	6	7	11	12

Dense Bitvector (DB)

Size [bits]: *n*

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 *n*



Set Representations

Input set

n = 16 (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

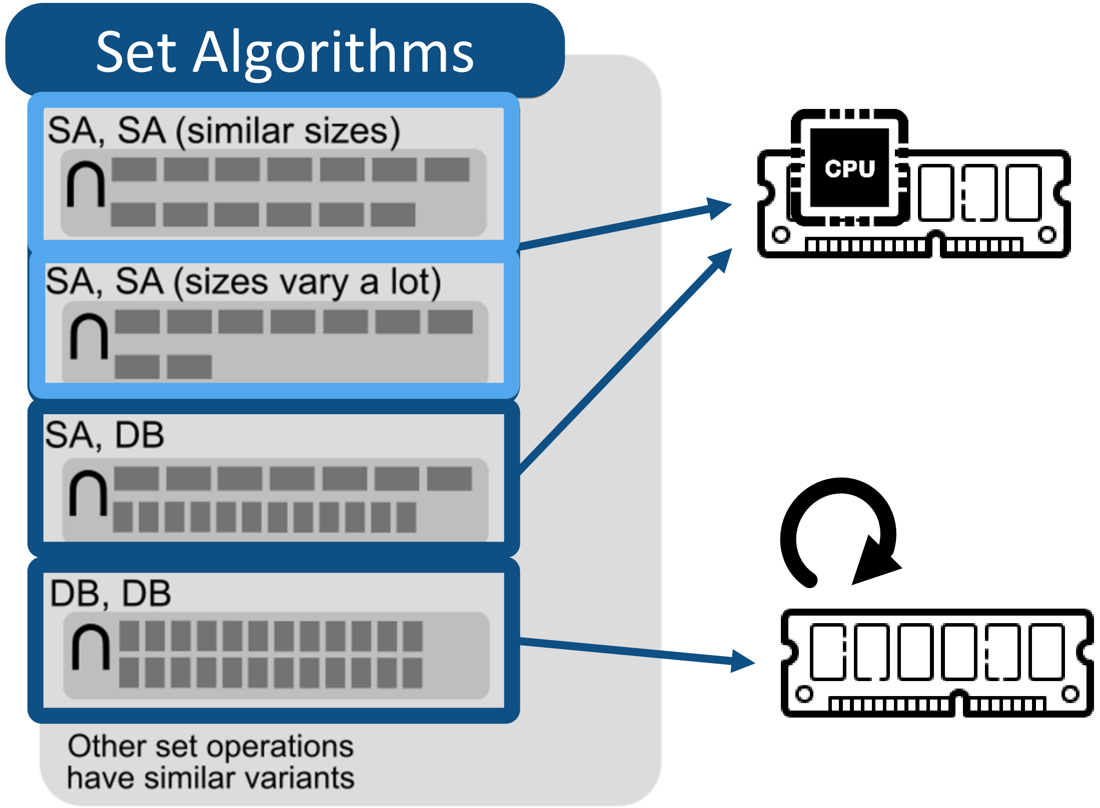
Size [bits]: *W* × *#vertices*

5	6	7	11	12

Dense Bitvector (DB)

Size [bits]: *n*

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 *n*



Set Representations

Input set

n = 16 (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

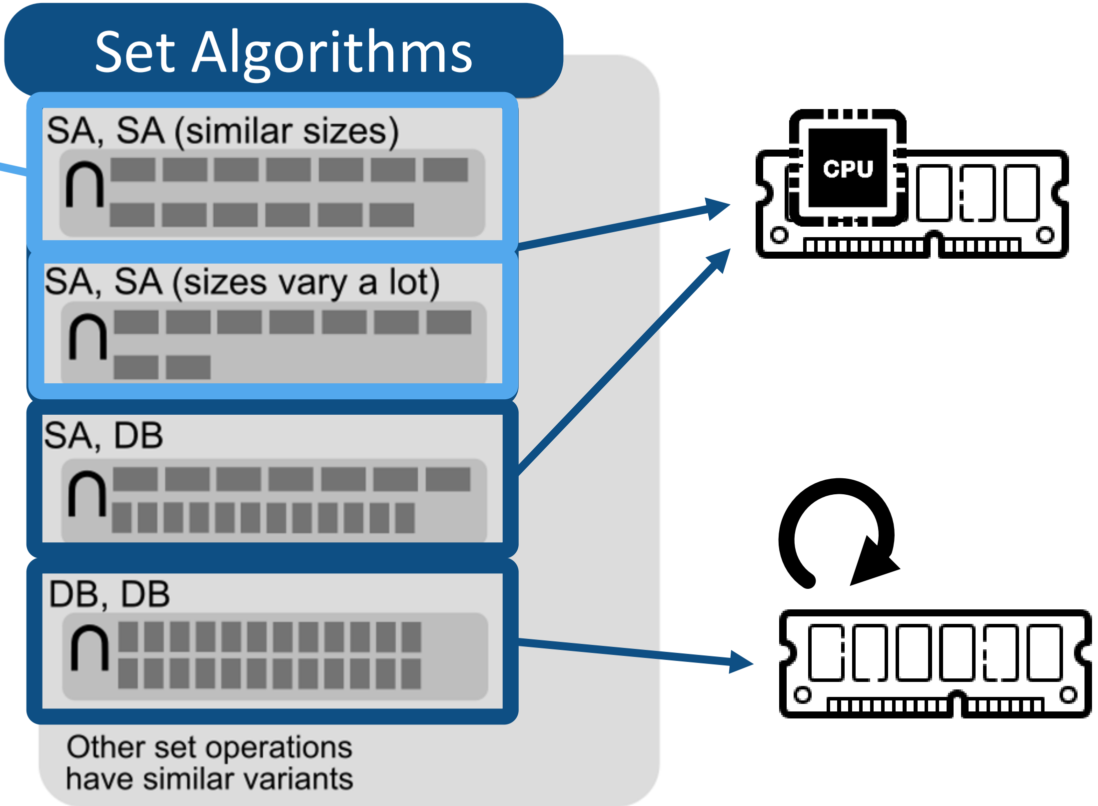
5	6	7	11	12

Dense Bitvector (DB)

Size [bits]: *n*

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 *n*

Merge: Iterate through two input sets (sorted), identifying common elements



Set Representations

Input set

n = *16* (#vertices)
{0, ..., 15}

An example set:
{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

5	6	7	11	12

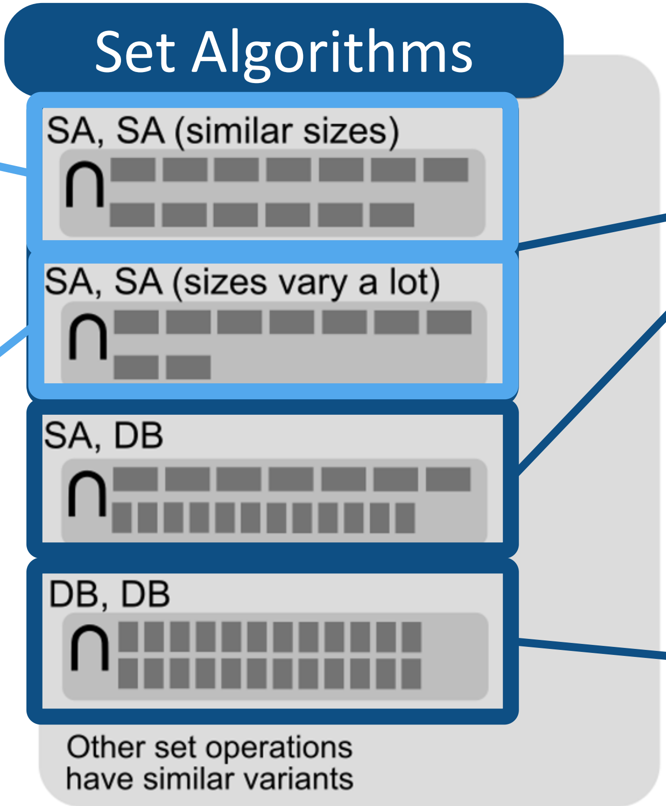
Dense Bitvector (DB)

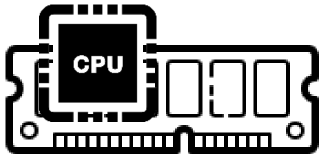
Size [bits]: *n*

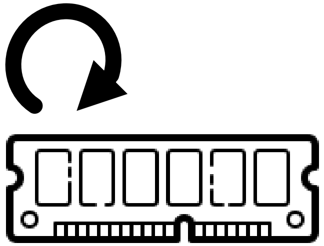
1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 *n*

Merge: Iterate through two input sets (sorted), identifying common elements

Galloping: iterate over the elements of a smaller set and use a binary search to check if each element is in the bigger set







Set Representations

Input set

n = 16 (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

5	6	7	11	12

Dense Bitvector (DB)

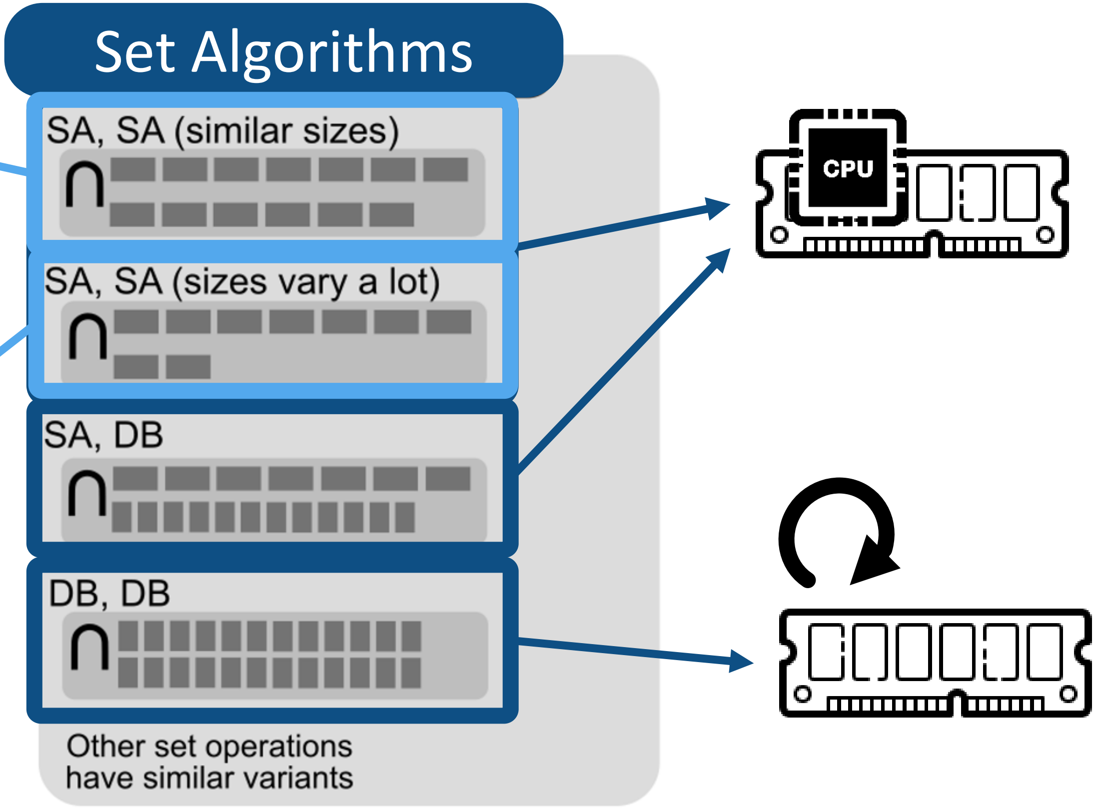
Size [bits]: *n*

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 *n*

Merge: Iterate through two input sets (sorted), identifying common elements

Complexity: O(n+m)

Galloping: iterate over the elements of a smaller set and use a binary search to check if each element is in the bigger set



Set Representations

Input set

n = *16* (#vertices)
{0, ..., 15}

An example set:
{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

| 5 | 6 | 7 | 11 | 12 |

Dense Bitvector (DB)

Size [bits]: *n*

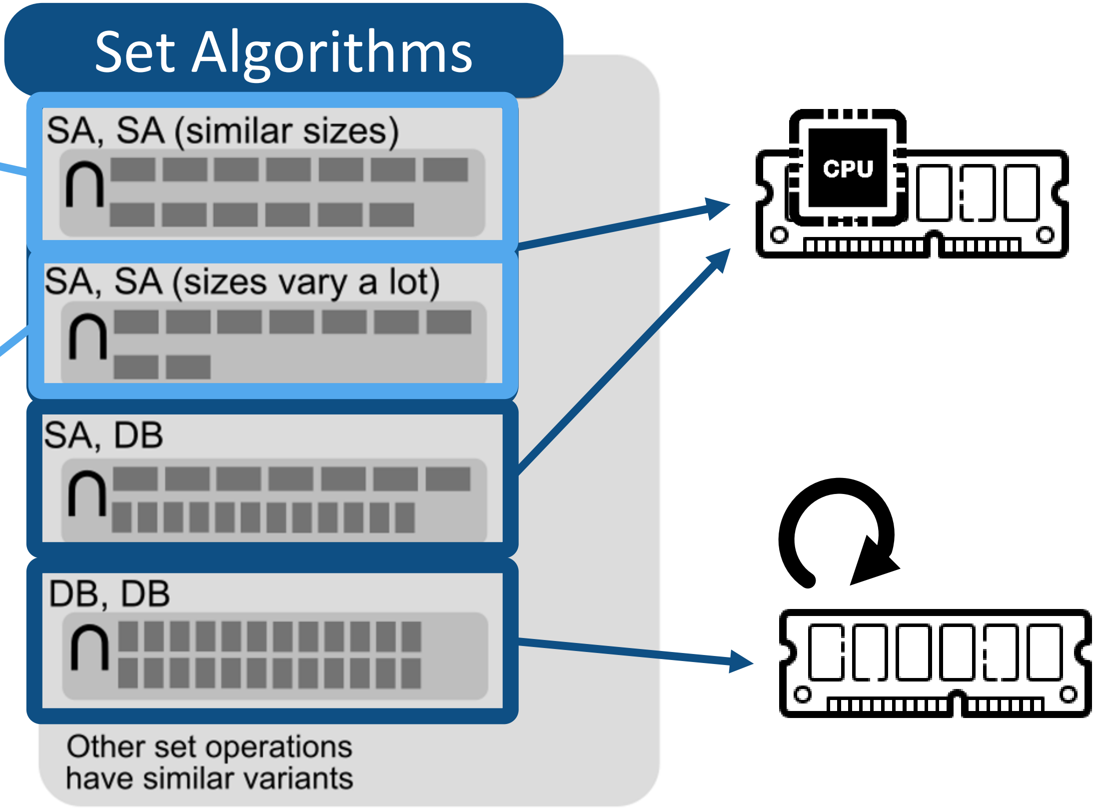
1 | 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 | *n*

Merge: Iterate through two input sets (sorted), identifying common elements

Complexity: O(n+m)

Galloping: iterate over the elements of a smaller set and use a binary search to check if each element is in the bigger set

Complexity: O(m log n)



Set Representations

SISA offers all these variants

Input set

$n = 16$ (#vertices)
{0, ..., 15}

An example set:
{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

5	6	7	11	12

Dense Bitvector (DB)

Size [bits]: *n*

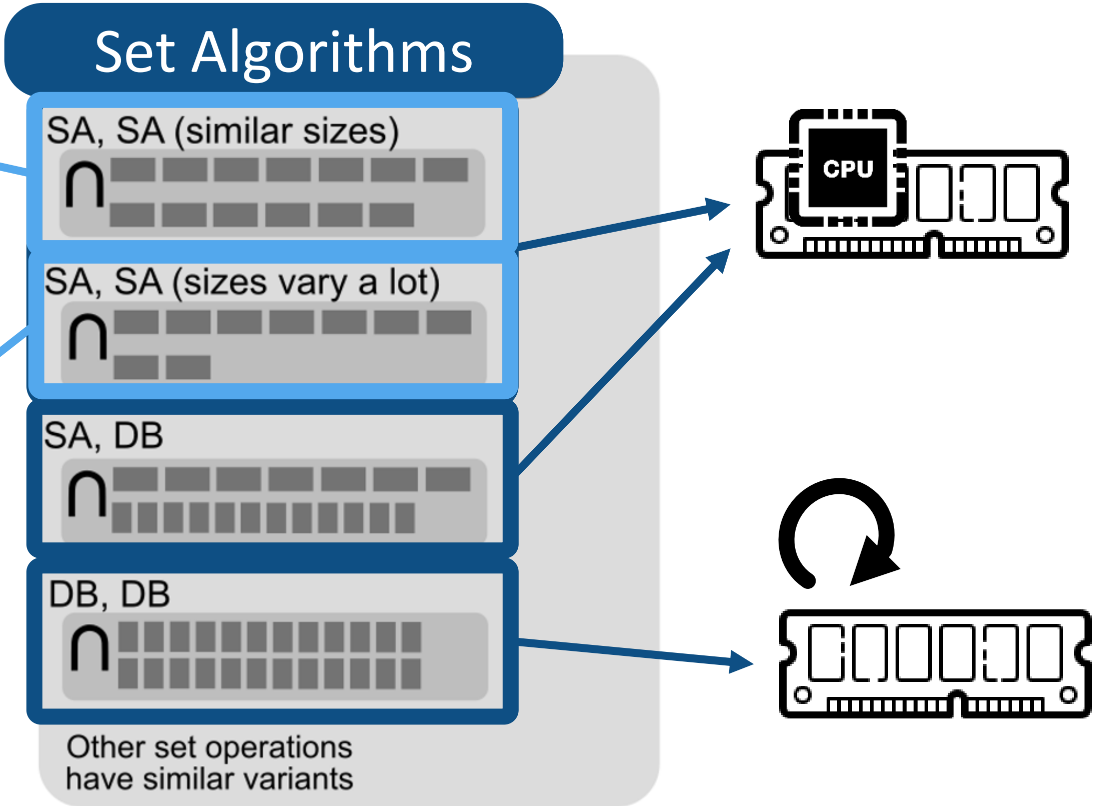
1 0000011100011000 *n*

Merge: Iterate through two input sets (sorted), identifying common elements

Complexity: O(n+m)

Galloping: iterate over the elements of a smaller set and use a binary search to check if each element is in the bigger set

Complexity: O(m log n)



Our design comes with...

1 ... **Set-centric paradigm & formulations** of *many* graph mining algorithms, coming with guarantees for **theoretical efficiency**

2 ... **Set-centric ISA** with **high-performance set organization**: set representations & set algorithms

3 ... **Hardware implementation of SISA** with processing-using-memory (SISA-PUM) and processing near memory (SISA-PNM)

Our design comes with...

1

... **Set-centric paradigm & formulations** of *many* graph mining algorithms, coming with guarantees for **theoretical efficiency**

2

... **Set-centric ISA** with **high-performance set organization**: set representations & set algorithms

3

... **Hardware implementation of SISA** with processing-using-memory (SISA-PUM) and processing near memory (SISA-PNM)

Our design comes with...

1

... **Set-centric paradigm & formulations** of *many* graph mining algorithms, coming with guarantees for **theoretical efficiency**

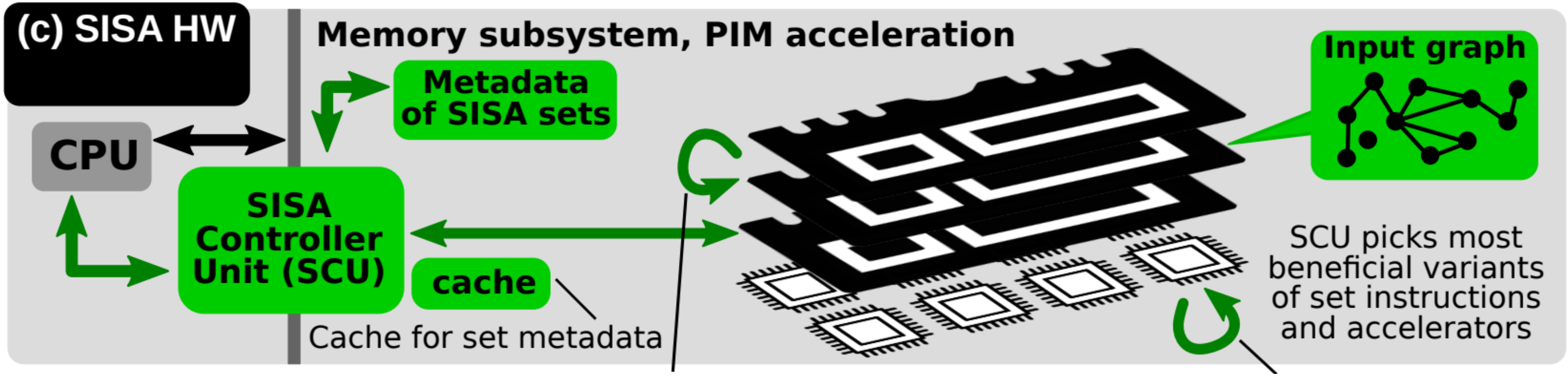
2

... **Set-centric ISA** with **high-performance set organization**: set representations & set algorithms

3

... **Hardware implementation of SISA** with processing-using-memory (SISA-PUM) and processing near memory (SISA-PNM)

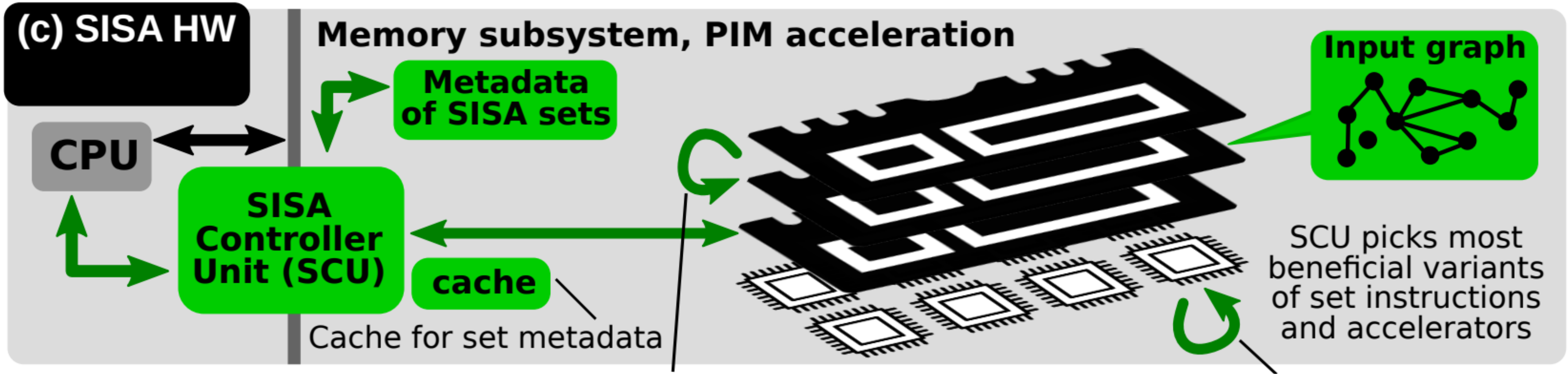
SISA Example Hardware Implementation



SISA-PUM (in-situ PIM, e.g., bulk bitwise operations with Ambit**)**

SISA-PNM (near-memory PIM, e.g., logic layers in 3D DRAM or 2D UPMEM**)**

SISA Example Hardware Implementation



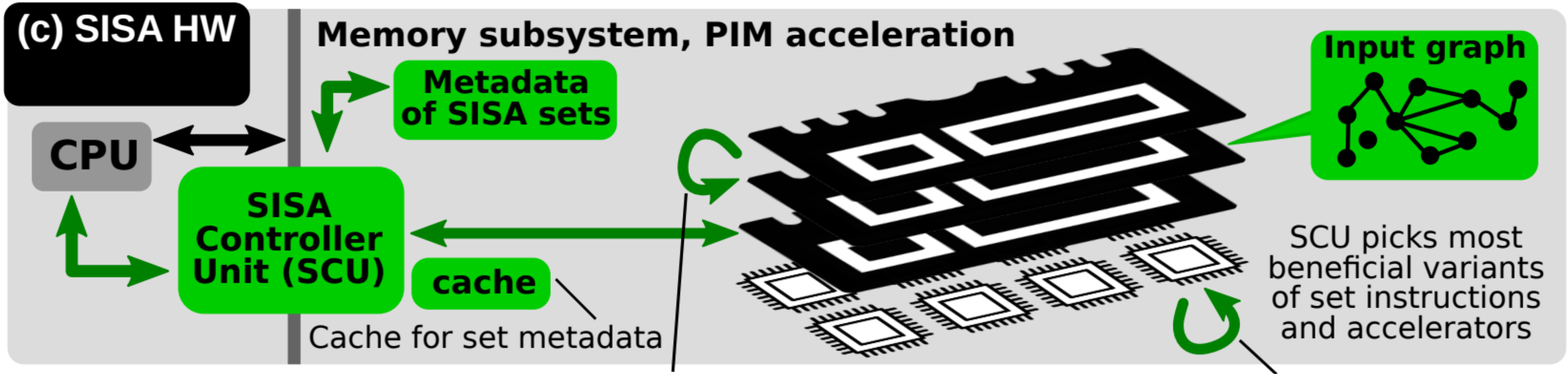
Green blocks: SISA related

SISA-PUM (in-situ PIM, e.g., bulk bitwise operations with Ambit**)**

SISA-PNM (near-memory PIM, e.g., logic layers in 3D DRAM or 2D UPMEM**)**

SISA Example Hardware Implementation

Execute a given set operation



Green blocks: SISA related

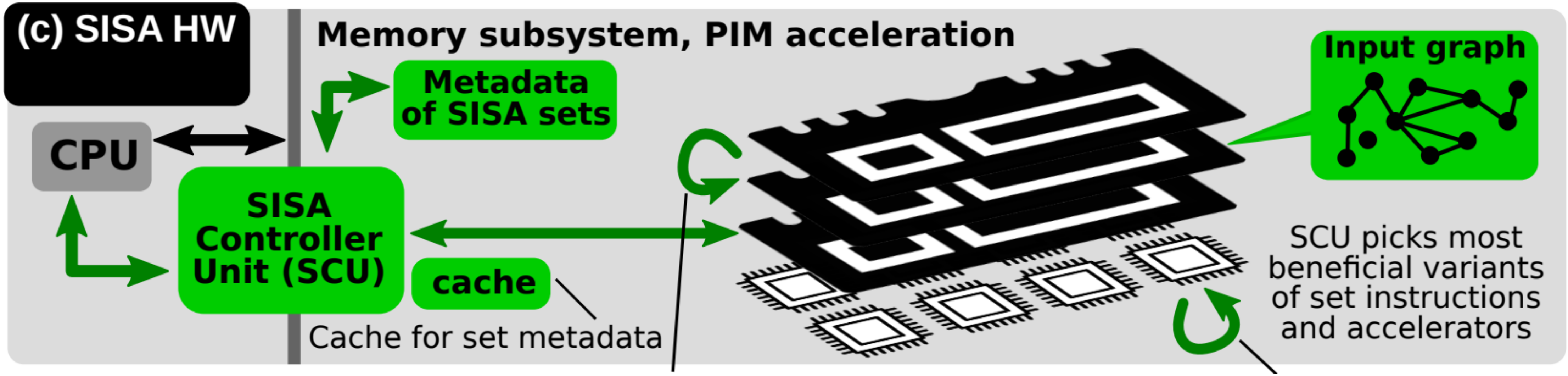
SISA-PUM (in-situ PIM, e.g., bulk bitwise operations with Ambit**)**

SISA-PNM (near-memory PIM, e.g., logic layers in 3D DRAM or 2D UPMEM**)**

SISA Example Hardware Implementation

Execute a given set operation

Select the best set operation variant



Green blocks: SISA related

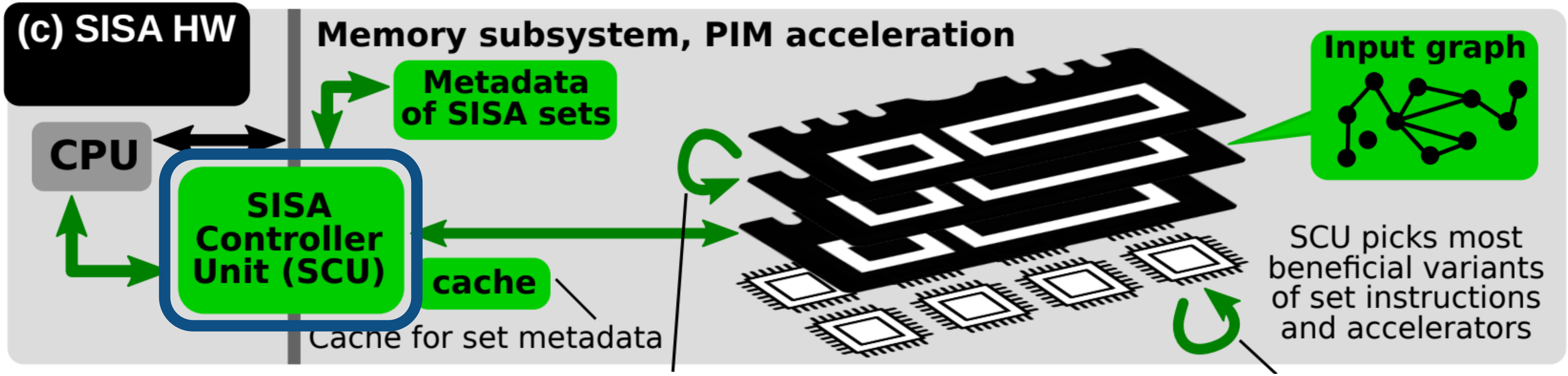
SISA-PUM (in-situ PIM, e.g., bulk bitwise operations with Ambit**)**

SISA-PNM (near-memory PIM, e.g., logic layers in 3D DRAM or 2D UPMEM**)**

SISA Example Hardware Implementation

Execute a given set operation

Select the best set operation variant



Green blocks: SISA related

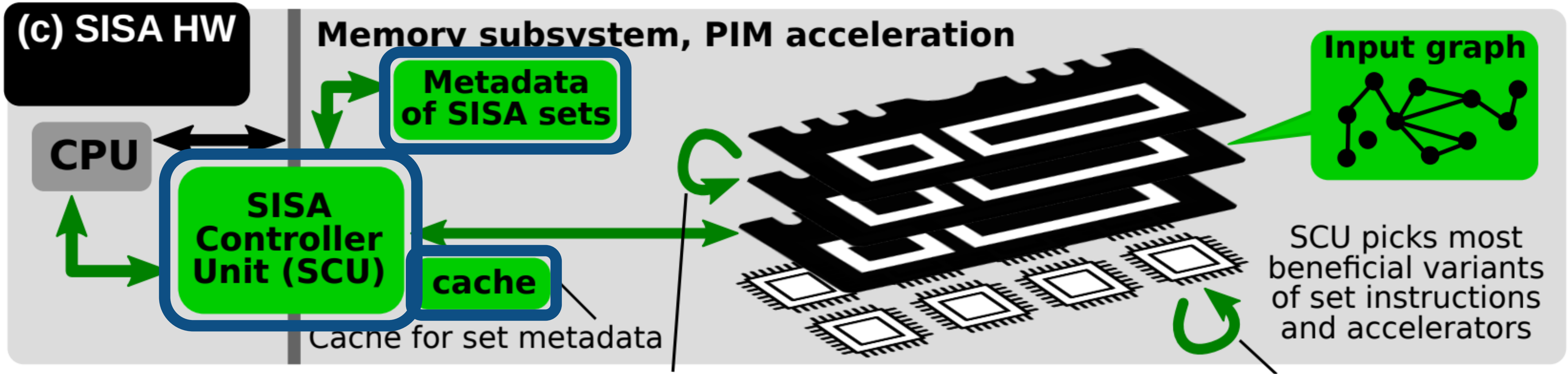
SISA-PUM (in-situ PIM, e.g., bulk bitwise operations with Ambit**)**

SISA-PNM (near-memory PIM, e.g., logic layers in 3D DRAM or 2D UPMEM**)**

SISA Example Hardware Implementation

Execute a given set operation

Select the best set operation variant



Green blocks: SISA related

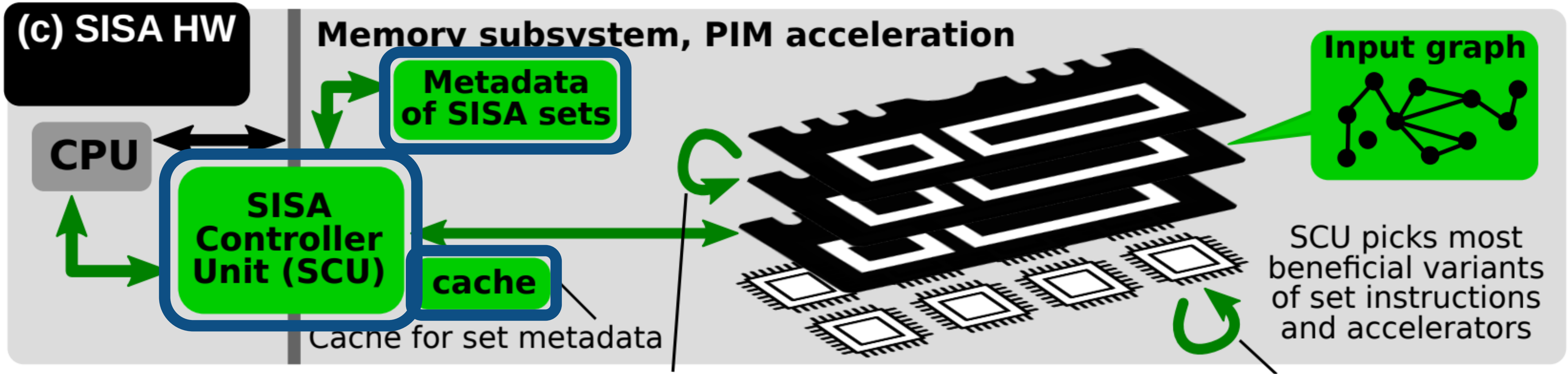
SISA-PUM (in-situ PIM, e.g., bulk bitwise operations with Ambit**)**

SISA-PNM (near-memory PIM, e.g., logic layers in 3D DRAM or 2D UPMEM**)**

SISA Example Hardware Implementation

Execute a given set operation

Select the best set operation variant



Green blocks: SISA related

SISA-PUM (in-situ PIM, e.g., bulk bitwise operations with Ambit**)**

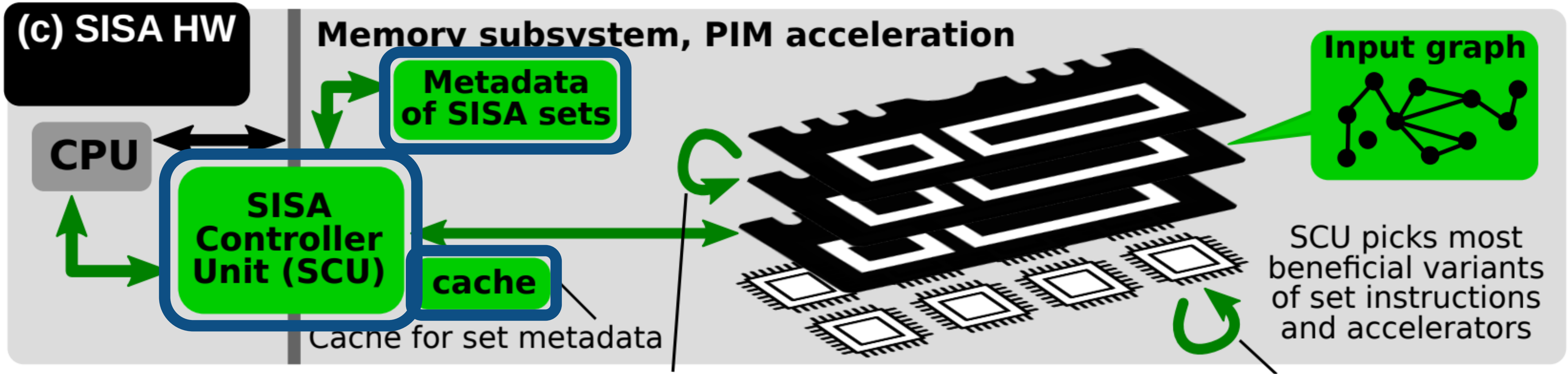
SISA-PNM (near-memory PIM, e.g., logic layers in 3D DRAM or 2D UPMEM**)**

Consider set representations (select SISA-PUM vs. SISA-PNM)

SISA Example Hardware Implementation

Execute a given set operation

Select the best set operation variant



Green blocks: SISA related

SISA-PUM (in-situ PIM, e.g., bulk bitwise operations with Ambit**)**

SISA-PNM (near-memory PIM, e.g., logic layers in 3D DRAM or 2D UPMEM**)**

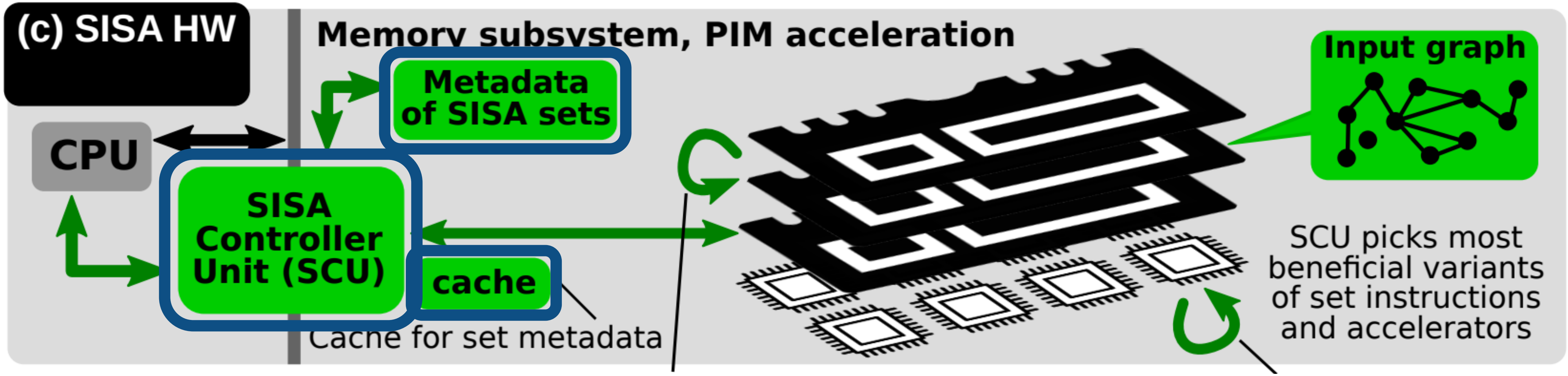
Consider set representations (select SISA-PUM vs. SISA-PNM)

Consider set operation variants (select Galloping vs. Merge)

SISA Example Hardware Implementation

Execute a given set operation

Select the best set operation variant



Green blocks: SISA related

SISA-PUM (in-situ PIM, e.g., bulk bitwise operations with Ambit**)**

SISA-PNM (near-memory PIM, e.g., logic layers in 3D DRAM or 2D UPMEM**)**

Consider set representations (select SISA-PUM vs. SISA-PNM)

Consider set operation variants (select Galloping vs. Merge)

Use performance models (streaming vs. random memory access)

Evaluation

Goals & Setup

Evaluation
Goals & Setup

Goal: SISA enables accelerating the state of the art

Evaluation

Goals & Setup

Goal: SISA enables accelerating the state of the art

Main baselines:
„**non-set**": state of the art,
„**set-based**": set-centric + standard HW,
„**sisa**": set-centric + PIM acceleration

Evaluation

Goals & Setup

Evaluation
Goals & Setup

Simulation Infrastructure: Sniper [1] (cycle-level) with the Pin frontend [2]

[1] W. Heirman et al., Sniper: Scalable and accurate parallel multi-core simulation. ACACES, 2012.
[2] C.-K. Luk et al., Pin: building customized program analysis tools with dynamic instrumentation. ACM SIGPLAN Notices, 2005.

Evaluation
Goals & Setup

Simulation Infrastructure: Sniper [1] (cycle-level) with the Pin frontend [2]

Considered platforms: (1) SISA, (2) a high-performance Out-of-Order manycore CPU

[1] W. Heirman et al., Sniper: Scalable and accurate parallel multi-core simulation. ACACES, 2012.
[2] C.-K. Luk et al., Pin: building customized program analysis tools with dynamic instrumentation. ACM SIGPLAN Notices, 2005.

Evaluation
Goals & Setup

Simulation Infrastructure: Sniper [1] (cycle-level) with the Pin frontend [2]

Considered platforms: (1) SISA, (2) a high-performance Out-of-Order manycore CPU

Parametrization: Such as in past work (Tessarect [3]); all baselines use bandwidth scalability

[1] W. Heirman et al., Sniper: Scalable and accurate parallel multi-core simulation. ACACES, 2012.
[2] C.-K. Luk et al., Pin: building customized program analysis tools with dynamic instrumentation. ACM SIGPLAN Notices, 2005.
[3] J. Ahn et al., A scalable processing-in-memory accelerator for parallel graph processing. ISCA, 2015.

Evaluation

Goals & Setup

Simulation Infrastructure: Sniper [1] (cycle-level) with the Pin frontend [2]

Considered platforms: (1) SISA, (2) a high-performance Out-of-Order manycore CPU

Parametrization: Such as in past work (Tessarect [3]); all baselines use bandwidth scalability

Problems: k-cliques, k-star-cliques, maximal cliques, clustering (using the Jaccard, overlap, and total neighbors as vertex similarity coefficients), subgraph isomorphism

[1] W. Heirman et al., Sniper: Scalable and accurate parallel multi-core simulation. ACACES, 2012.
[2] C.-K. Luk et al., Pin: building customized program analysis tools with dynamic instrumentation. ACM SIGPLAN Notices, 2005.
[3] J. Ahn et al., A scalable processing-in-memory accelerator for parallel graph processing. ISCA, 2015.

Evaluation
Goals & Setup

Simulation Infrastructure: Sniper [1] (cycle-level) with the Pin frontend [2]

Considered platforms: (1) SISA, (2) a high-performance Out-of-Order manycore CPU

Parametrization: Such as in past work (Tessarect [3]); all baselines use bandwidth scalability

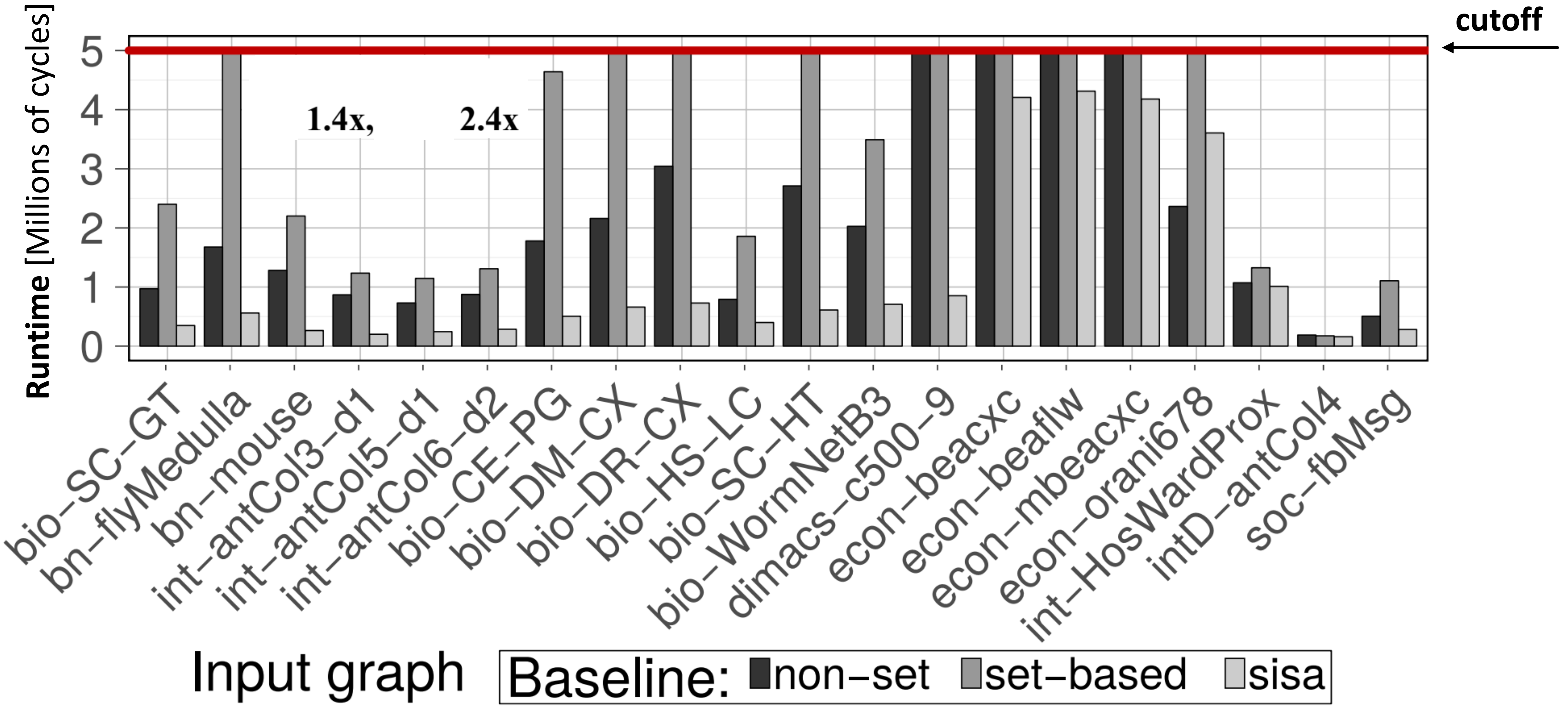
Graphs: biological (**bio-**), interaction (**int-**), social (**soc-**), brain (**bn-**), dynamic (**D**), web (**web-**), economical (**econ-**), and structural (**str-**) networks

Problems: k-cliques, k-star-cliques, maximal cliques, clustering (using the Jaccard, overlap, and total neighbors as vertex similarity coefficients), subgraph isomorphism

[1] W. Heirman et al., Sniper: Scalable and accurate parallel multi-core simulation. ACACES, 2012.
[2] C.-K. Luk et al., Pin: building customized program analysis tools with dynamic instrumentation. ACM SIGPLAN Notices, 2005.
[3] J. Ahn et al., A scalable processing-in-memory accelerator for parallel graph processing. ISCA, 2015.

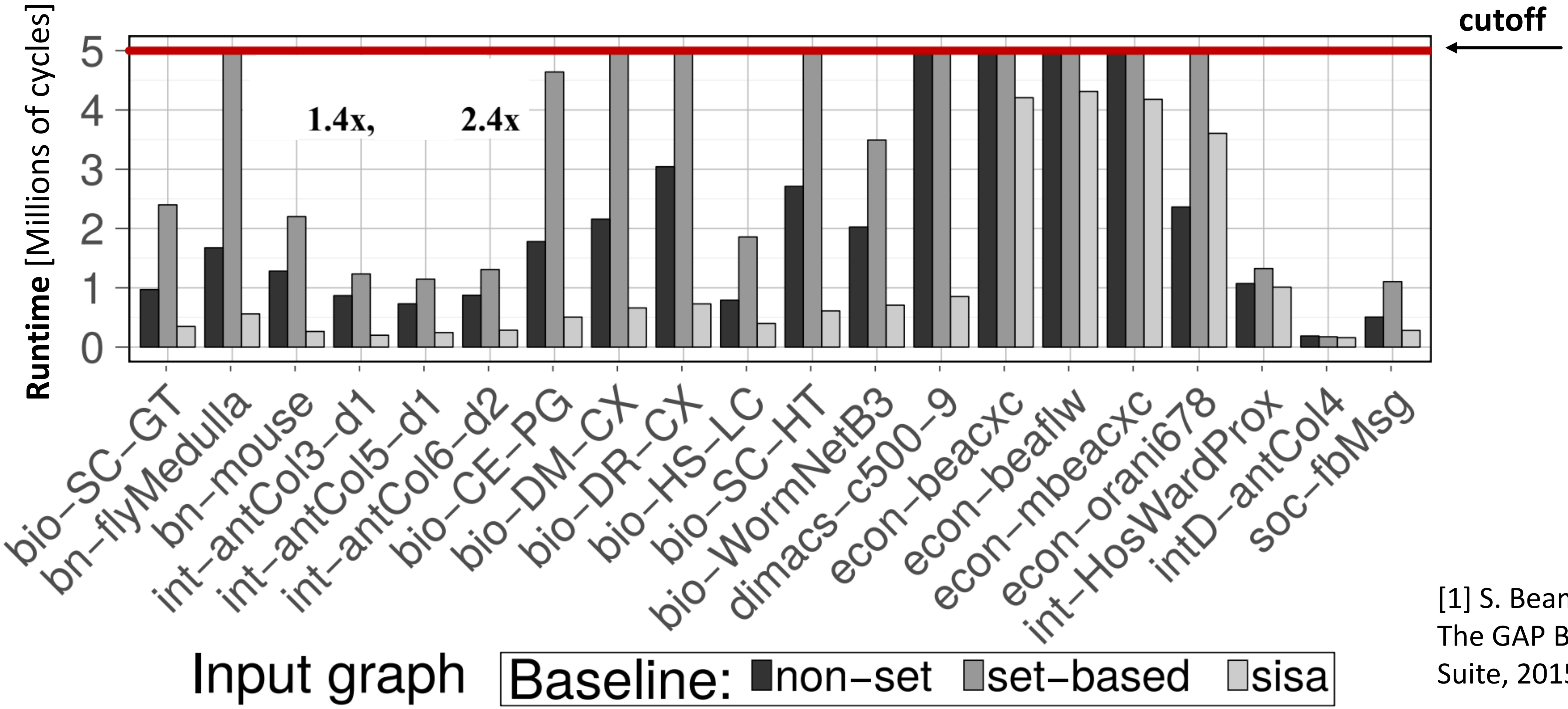
Pattern Matching: Clustering (Jaccard based) [1]

Cores/threads: 32



Pattern Matching: Clustering (Jaccard based) [1]

Cores/threads: 32

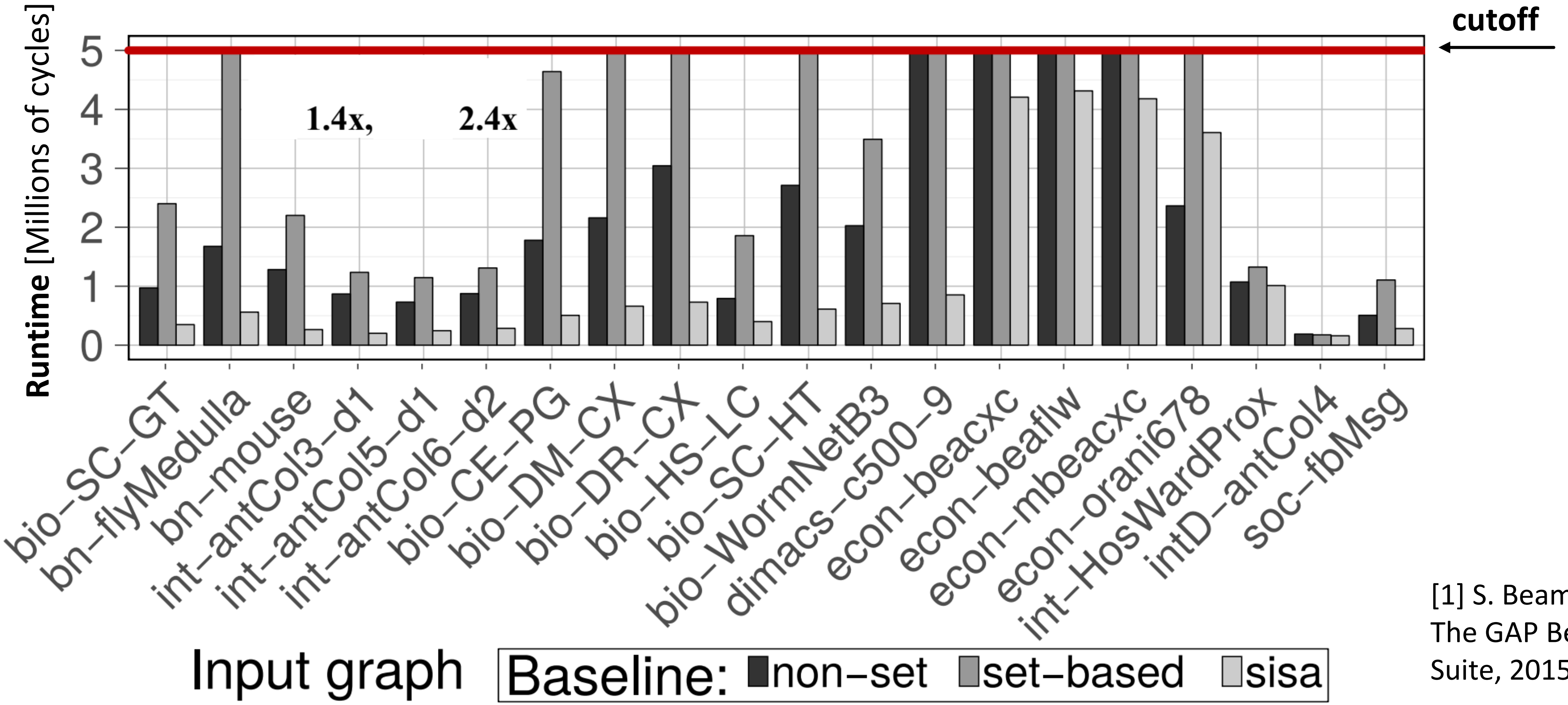


[1] S. Beamer et al., The GAP Benchmark Suite, 2015.

Pattern Matching: Clustering (Jaccard based) [1]

Cores/threads: 32

Complexity: $O(n^3)$



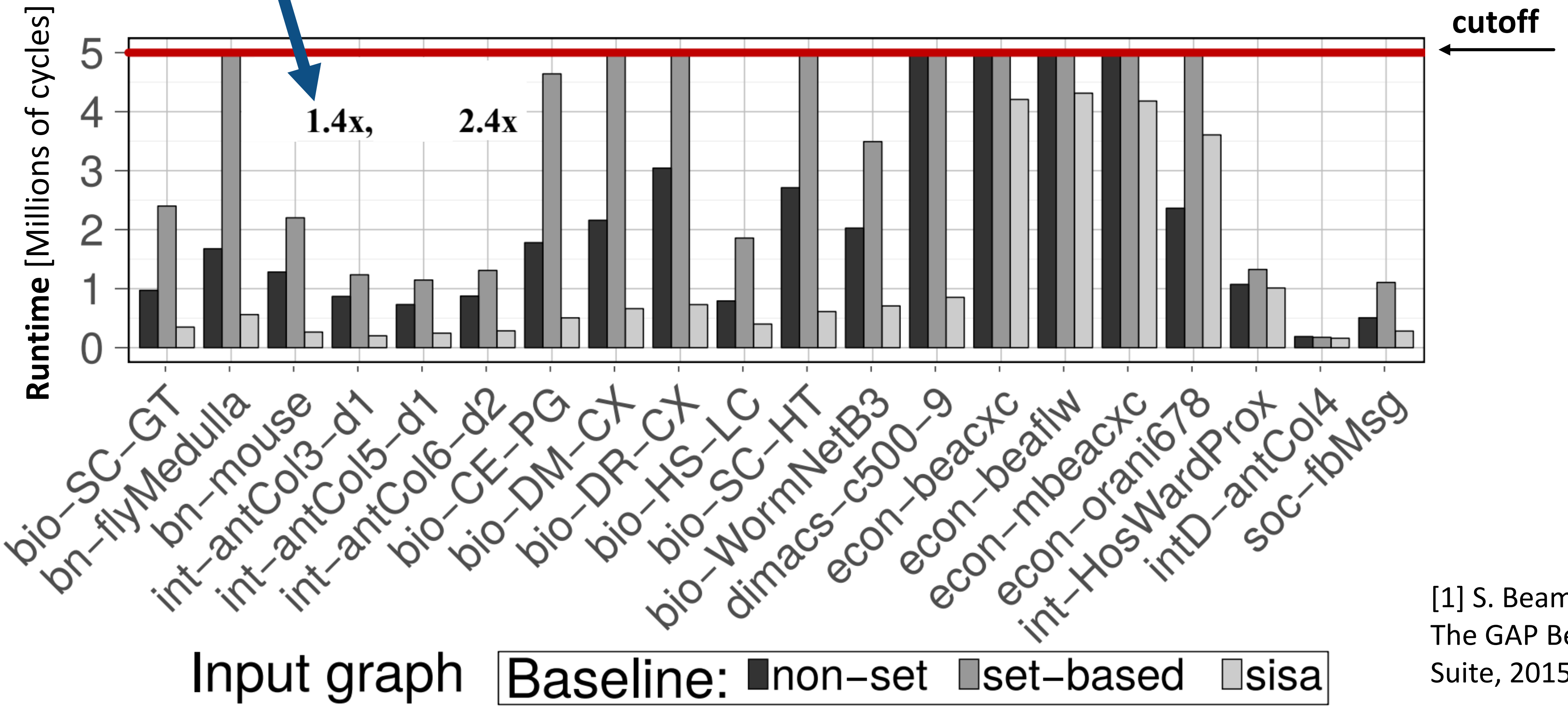
[1] S. Beamer et al., The GAP Benchmark Suite, 2015.

Pattern Matching: Clustering (Jaccard based) [1]

Cores/threads: 32

Complexity: O(n^3)

SISA's speedup over non-set



[1] S. Beamer et al., The GAP Benchmark Suite, 2015.

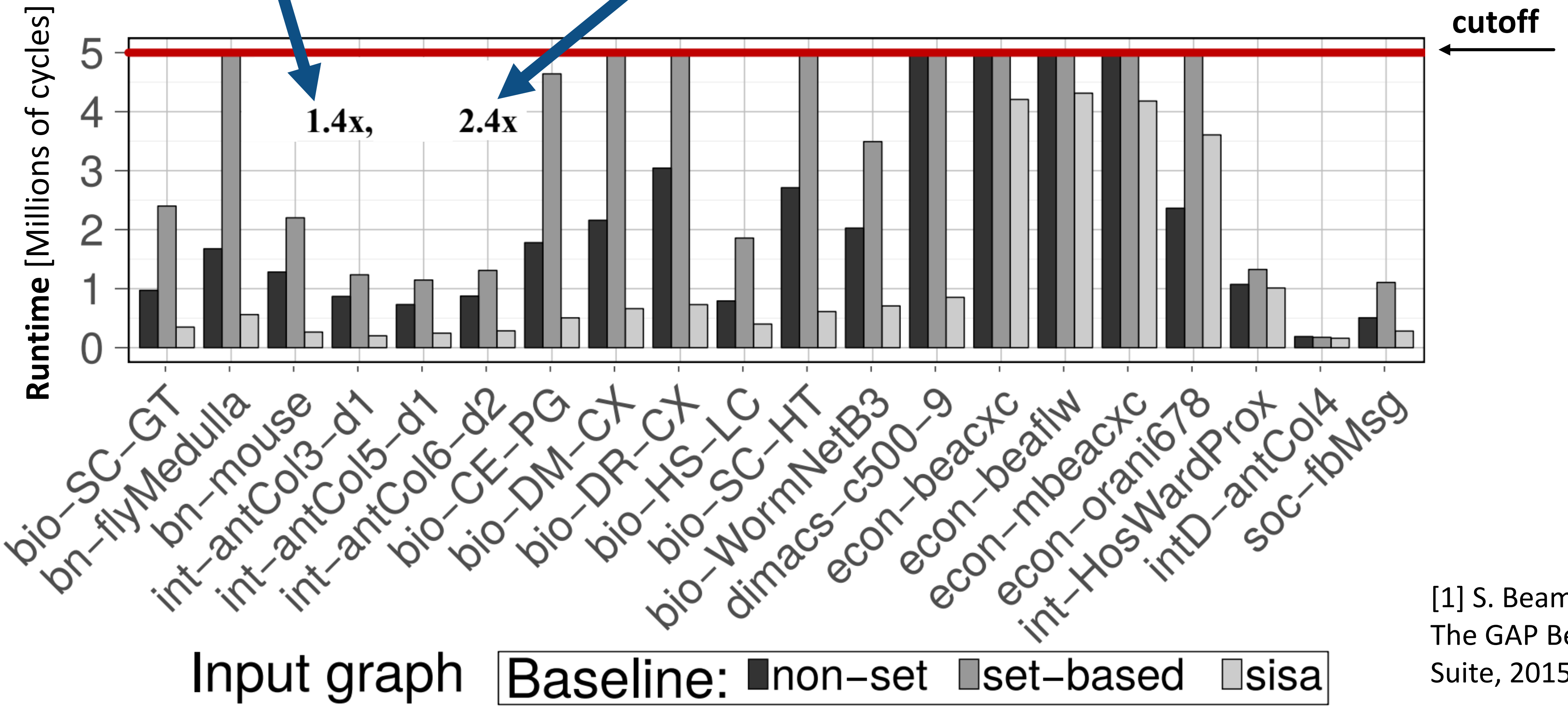
Pattern Matching: Clustering (Jaccard based) [1]

Cores/threads: 32

Complexity: O(n^3)

SISA's speedup over non-set

SISA's speedup over set-based



[1] S. Beamer et al., The GAP Benchmark Suite, 2015.

Pattern Matching: k-Cliques

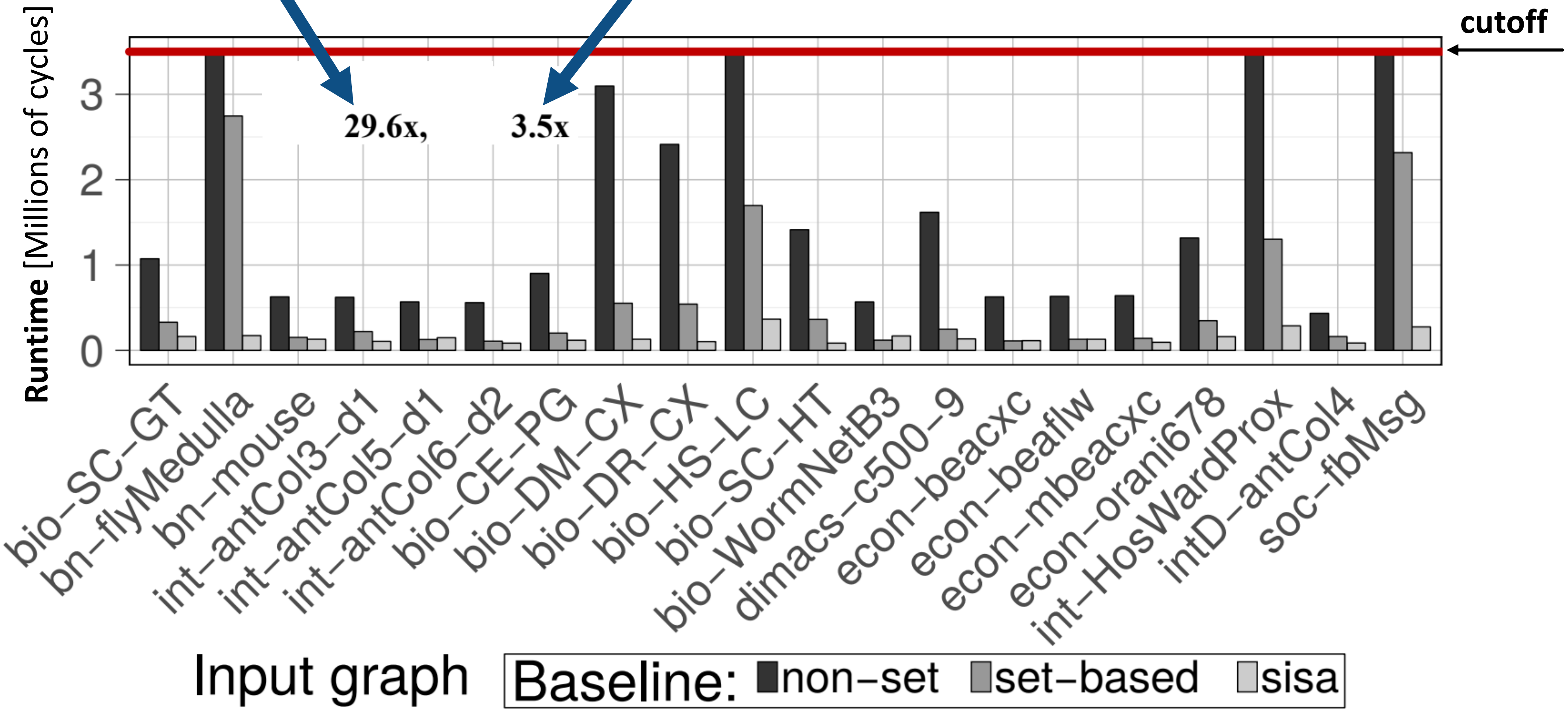
Cores/threads: 32

Complexity: $O(n^k)$

k = 5

SISA's speedup over non-set

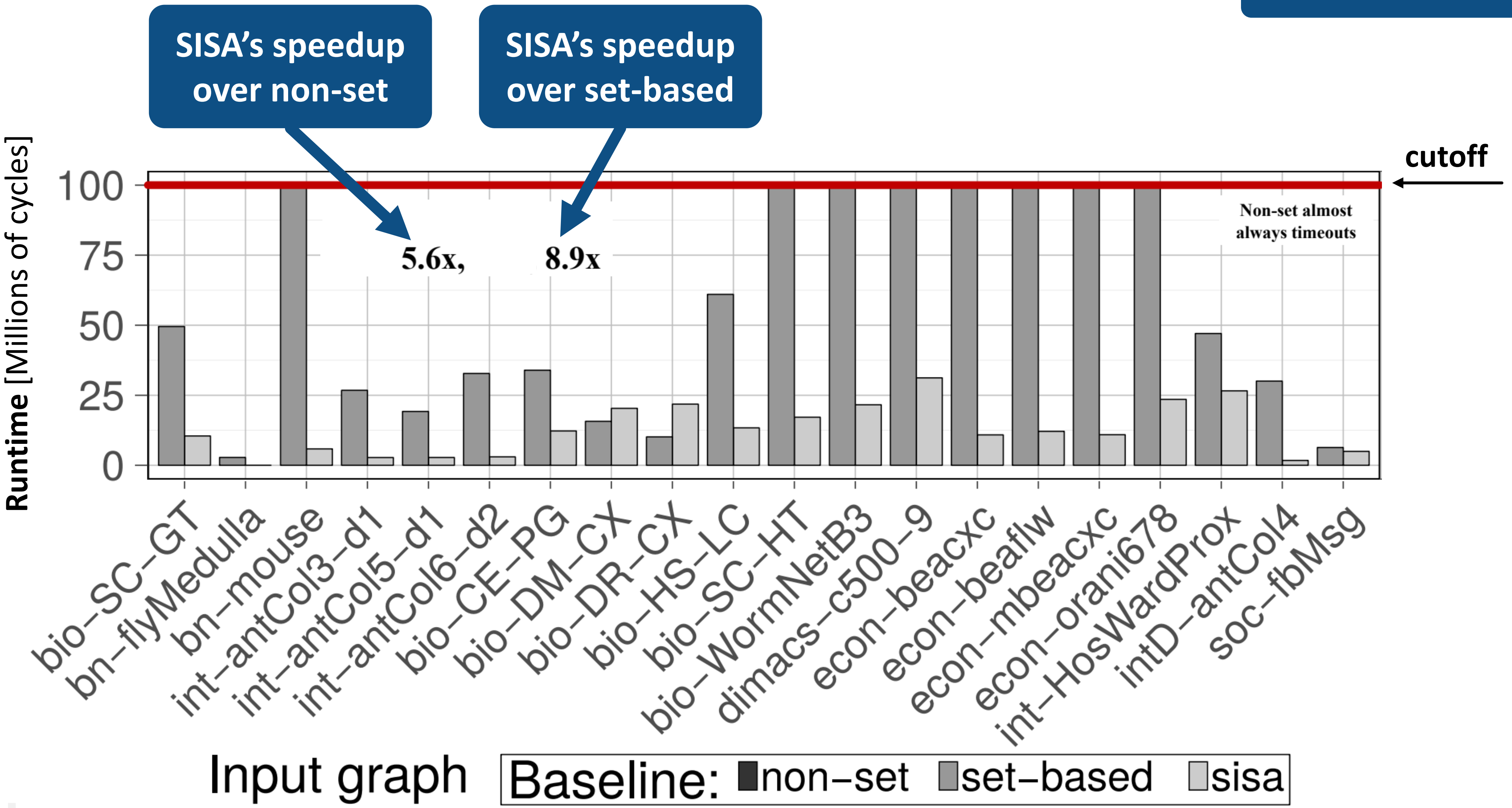
SISA's speedup over set-based

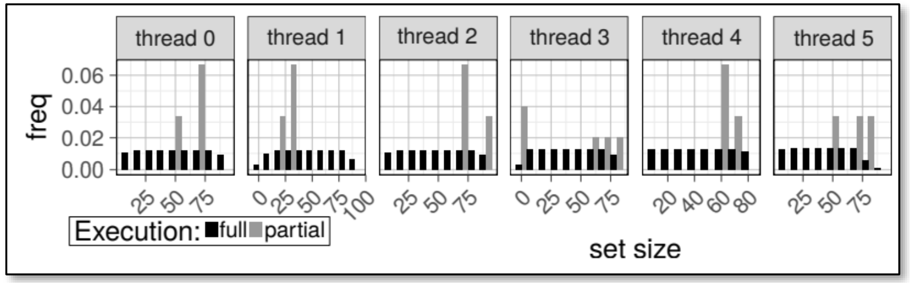


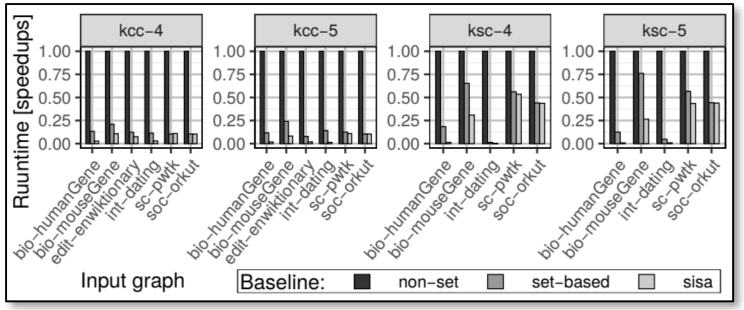
Pattern Matching: Maximal Cliques

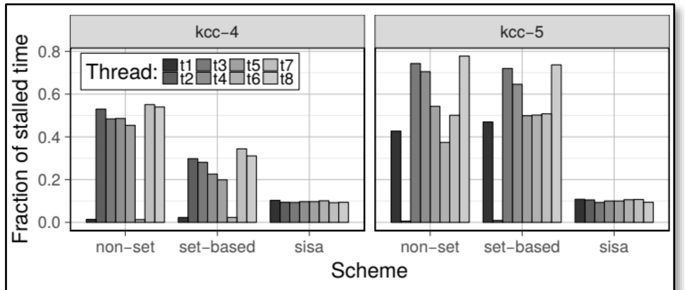
Cores/threads: 32

Complexity: $O(3^{n/3})$

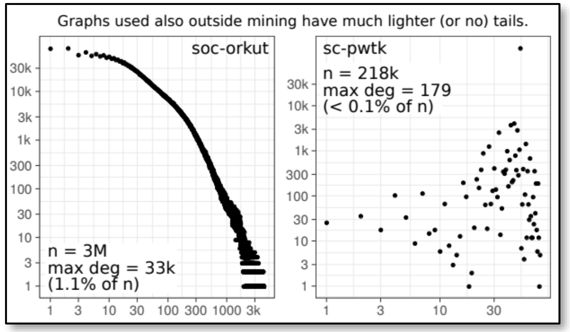


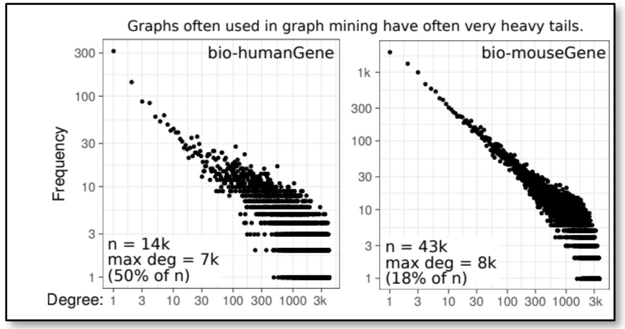


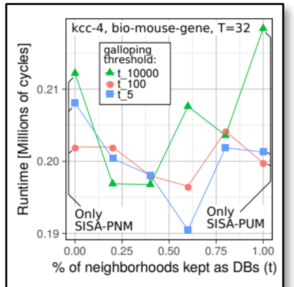




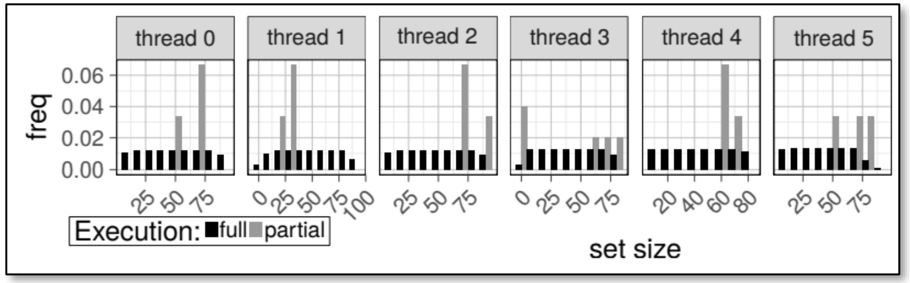
(a) Total amounts of stall times of different 8 parallel threads.

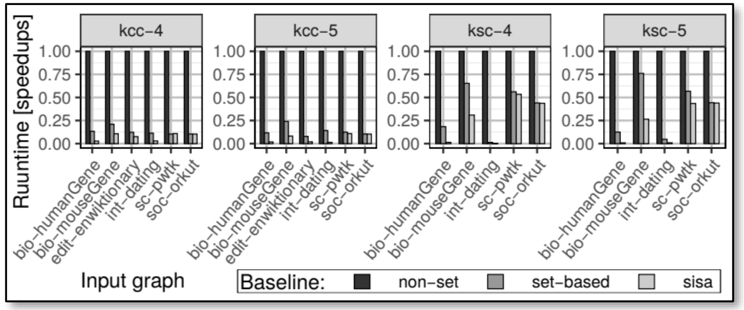


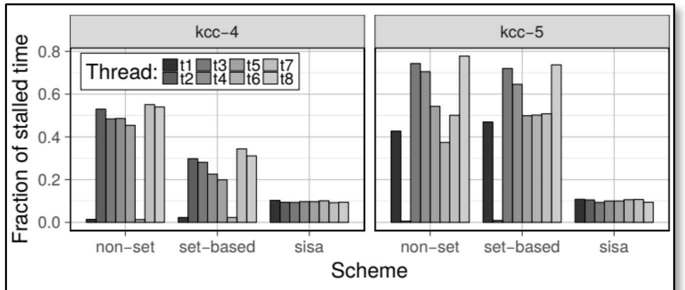




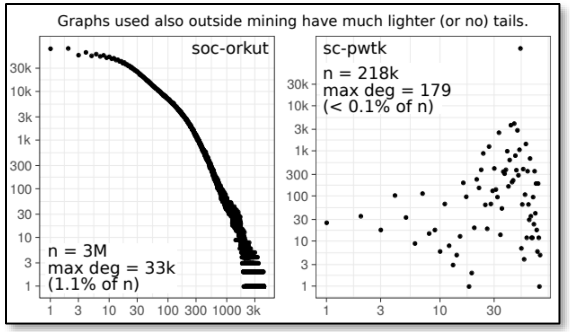
(b) Sensitivity analysis.

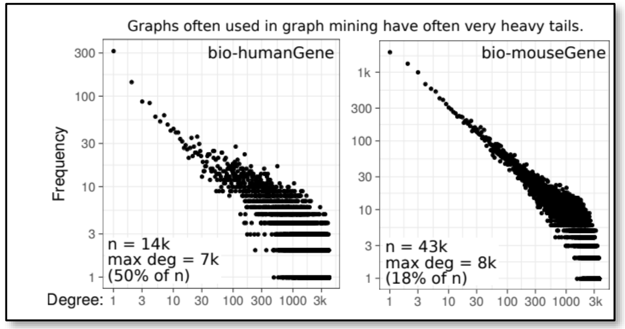


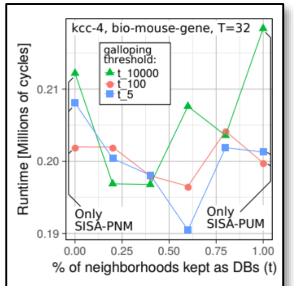




(a) Total amounts of stall times of different 8 parallel threads.

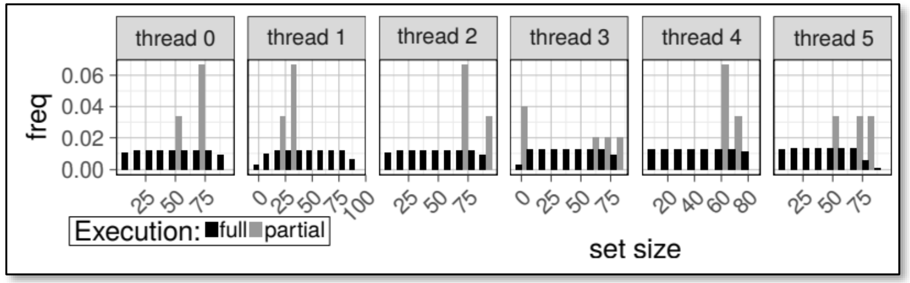


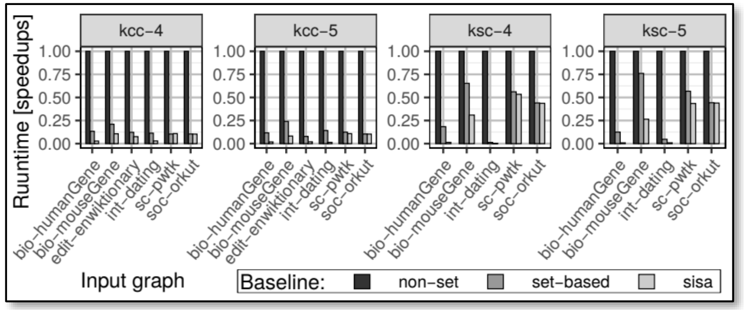


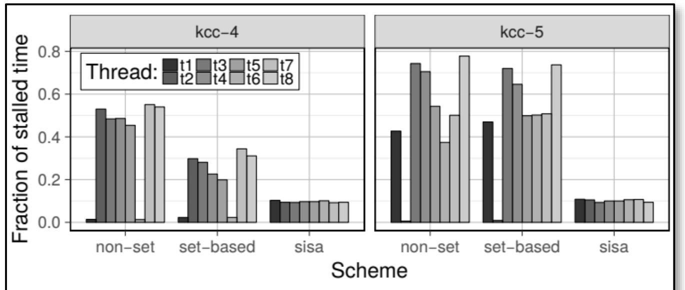


(b) Sensitivity analysis.

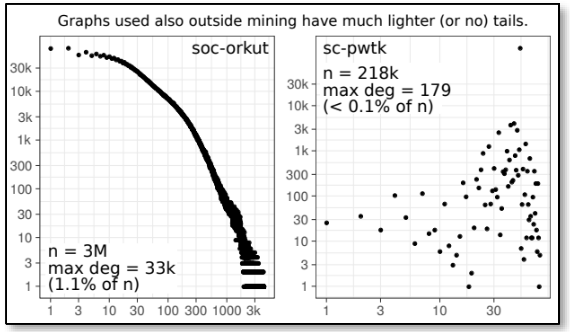
	Triangle Counting [158]	k-Clique Listing [53]	k-Star-Clique Listing [84]	Maximal Cliques Listing [33, 62]	Link Prediction†	Link Prediction‡	Link Prediction§	Jarvis-Patrick Clustering [86]
SISA + merging intersection	$O(mc)$★	$O\left(km\left(\frac{c}{2}\right)^{k-2}\right)$★	$O\left(k^2m\left(\frac{c}{2}\right)^{k-1}\right)$★	$O\left(cdn3^{c/3}\right)$	$O(md)$	$O\left(n^2+md\right)$	$O\left(n^2\right)$★	$O(md)$
SISA + galloping intersection	$O(mc\log c)$	$O\left(km\left(\frac{c}{2}\right)^{k-2}\log c\right)$	$O\left(k^2m\left(\frac{c}{2}\right)^{k-1}\log c\right)$	$O\left(cn3^{c/3}\right)$★	$O(mc\log c)$★	$O\left(n^2+mc\log c\right)$★	$O\left(n^2\right)$★	$O(mc\log d)$★

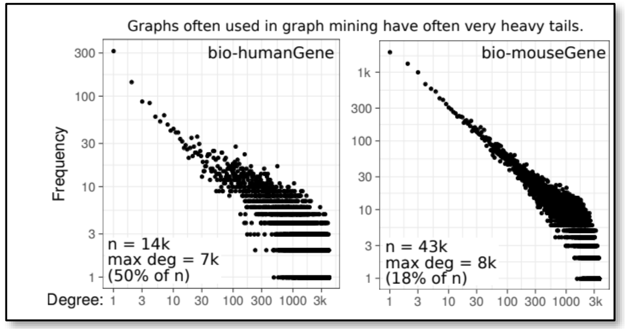


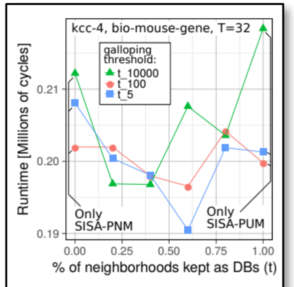




(a) Total amounts of stall times of different 8 parallel threads.







(b) Sensitivity analysis.

	Triangle Counting [158]	k-Clique Listing [53]	k-Star-Clique Listing [84]	Maximal Cliques Listing [33, 62]	Link Prediction†	Link Prediction‡	Link Prediction§	Jarvis-Patrick Clustering [86]
SISA + merging intersection	$O(mc)$★	$O\left(km\left(\frac{c}{2}\right)^{k-2}\right)$★	$O\left(k^2m\left(\frac{c}{2}\right)^{k-1}\right)$★	$O\left(cdn3^{c/3}\right)$	$O(md)$	$O\left(n^2+md\right)$	$O\left(n^2\right)$★	$O(md)$
SISA + galloping intersection	$O(mc\log c)$	$O\left(km\left(\frac{c}{2}\right)^{k-2}\log c\right)$	$O\left(k^2m\left(\frac{c}{2}\right)^{k-1}\log c\right)$	$O\left(cn3^{c/3}\right)$★	$O(mc\log c)$★	$O\left(n^2+mc\log c\right)$★	$O\left(n^2\right)$★	$O(mc\log d)$★

Reference / Accelerator	Prob.	Key memory mechanism
[Pi] GaaS-X [38]	SpMV	[e] CAM/MAC
[Pi] GraphSAR [52]	ver-c	[e] ReRAM
[Pi] GraphiDe [10]	low-c	[e] DRAM
[Pi] GraphIA [106]	edge-c	[e] DRAM
[Pc] Spara [200]	ver-c	[e] ReRAM
[Pc] GraphQ [209]	ver-c	[e] HMC
[Pc] GraphS [11]	low-c	[e] SOT-MRAM
[Pc] RAGra [82]	ver-c	[e] 3D ReRAM
[Pc] GRAM [201]	ver-c	[e] ReRAM
[Pc] GraphR [162]	SpMV	[e] ReRAM
[Pc] GraphP [196]	ver-c	[e] HMC
[Pc] Tesseract [5]	low-c	[e] HMC
[Pc] PIM-Enabled [6]	low-c	[e] HMC
[Pc] Gao et al. [66]	low-c	3D DRAM
[Pc] LiM [207, 208]	SpMSpM	[e] 3D DRAM
[A] Gramer [94]	pattern m.	DRAM, cache
[A] TrieJax [94]	joins	DRAM, LLC
[A] HyGCN [189]	GCN	eDRAM
[A] Outerspace [136]	SpMSpM	HBM
[A] Domino [184]	low-c	on-chip buffers
[A] GraphPIM [131]	low-c	[e] HMC
[A] Graphicionado [73]	ver-c	[e] eDRAM
[A] Ozdal et al. [135]	ver-c	[e] caches
[M] GraphSSD [119]	low-c	[e] SSD
[M] GRASP [63]	low-c	[e] LLC
[M] DROPLET [12]	edge-c	[e] DRAM pref.
[M] Ainsworth [7]	low-c	[e] DRAM pref.
[M] HyVE [81]	ver-c	ReRAM, SRAM
[M] HATS [127]	low-c	[e] caches
[M] OSCAR [159]	edge-c	[e] scratchpads
[M] IMP [195]	low-c	[e] caches
[F] GraphABCD [191]	low-c	DRAM
[F] Wang et al. [175]	clustering	BRAM
[F] ForeGraph [49, 50]	low-c	BRAM
[F] Yang [190]	ver-c	DRAM
[F] Yao [192]	low-c	DRAM
[F] Zhou [204]	edge-c	DRAM
[F] ExtraV [102]	low-c	DRAM
[F] Ma [116]	low-c	DRAM
[F] Zhou [205]	ver-c, edge-c	DRAM
[F] GraVF [61]	ver-c	BRAM
[F] Zhou [202, 203]	edge-c	DRAM
[F] GraphOps [134]	low-c	BRAM
[F] FPGP [48]	ver-c	DRAM
[F] GraphSoC [95]	low-c, SpMV	BRAM
[F] GraphGen [133]	ver-c	DRAM
[F] GraphStep [96]	low-c	BRAM
[F] Betkaoui et al. [30]	low-c	DRAM
[A+Pc] EnGN [75]	GNN	[e] HBM
[A+Pc] OMEGA [2]	low-c	[e] Scratchpads
[A+Pc+M] GraphH [51]	ver-c	[e] HMC
[F+Pc] HRL [67]	ver-c	[e] 3D DRAM
[Pc+Pi] SISA [This work]	Graph mining	PIM

SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems

MACIEJ BESTA, RAGHAVENDRA KANAKAGIRI, GRZEGORZ KWASNIEWSKI, RACHATA AUSAVARUNGNIRUN, JAKUB BERÁNEK, KONSTANTINOS KANELLOPOULOS, KACPER JANDA, ZUR VONARBURG-SHMARIA, LUKAS GIANINAZZI, IOANA STEFAN, JUAN GÓMEZ LUNA, MARCIN COPIK, LUKAS KAPP-SCHWOERER, SALVATORE DI GIROLAMO, MAREK KONIECZNY, ONUR MUTLU, TORSTEN HOEFLER

SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems

Maciej Besta, Raghavendra Kanakagiri, Grzegorz Kwasniewski, Rachata Ausavarungnirun, Jakub Beránek, Konstantinos Kanellopoulos, Kacper Janda, Zur Vonarburg-Shmaria, Lukas Gianinazzi, Ioana Stefan, Juan Gómez Luna, Marcin Copik, Lukas Kapp-Schwoerer, Salvatore Di Girolamo, Marek Konieczny, Onur Mutlu, Torsten Hoefler

Thank you for your attention

SPCL