SynCron
Efficient Synchronization Support for Near-Data-Processing Architectures

Christina Giannoula
Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas
Ivan Fernandez, Juan Gómez Luna, Lois Orosa
Nectarios Koziris, Georgios Goumas, Onur Mutlu
Near-Data-Processing (NDP) Systems

Graph Analytics

Recommendation Systems

Neural Networks

Bioinformatics
Synchronization is Necessary

**Single Source Shortest Path (SSSP)**

```python
for v in Graph:
    for u in neighbors[v]:
        if distance[v] + edge_weight[v, u] < distance[u]:
            lock_acquire(u)
            if distance[v] + edge_weight[v, u] < distance[u]:
                distance[u] = distance[v] + edge_weight[v, u]
            lock_release(u)
```

- **Graph Analytics**
- **Bioinformatics**
- **Databases**
- **Image Processing**
- **Bioinformatics**
- **Databases**
- **Image Processing**

**SAFARI**
Challenge: Efficient Synchronization

**NDP Unit**

- NDP Core
- NDP Core
- NDP Core

**Main Memory**

**NDP Unit**

- NDP Core
- NDP Core

**Programmable Core / Accelerator**

**Private Cache**

- Expensive Communication
- No Shared Caches

- No Hardware Cache Coherence
SynCron

The first end-to-end synchronization solution for NDP architectures

SynCron’s Benefits:

1. High System Performance
2. Low Hardware Cost
3. Programming Ease
4. General Synchronization Support
Outline

NDP Synchronization Solution Space

Our Mechanism: SynCron

Evaluation
Synchronization challenges in NDP systems:

1. Lack of hardware cache coherence support
2. Expensive communication across NDP units
3. Lack of a shared level of cache memory
NDP Synchronization Solution Space

(1) Shared Memory
- Hardware Cache Coherence
- Remote Atomics
- Specialized Hardware Support

(2) Message-passing
- Software-based Schemes
- Specialized Hardware Support
TCP Synchronization Solution Space

(1) Shared Memory
- Hardware Cache Coherence
- Remote Atomics
- Specialized Hardware Support

(2) Message-passing
- Software-based Schemes
- Specialized Hardware Support

Lack of hardware cache coherence support
NDP Synchronization Solution Space

(1) Shared Memory
- Hardware Cache Coherence
- Remote Atomics
- Specialized Hardware Support
  - CPUs: SSB [ISCA'07], Lock Cache [CASES'01]
  - MPPs: Full/Empty Bits [ISCA'83]
  - GPUs: Fermi GF100 [IEEE Micro'10], SGI Origin [ISCA'97], Cray T3E [ASPLOS'96]

(2) Message-passing
- Software-based Schemes
  - CPUs: MiSAR [ISCA'15], LCU [MICRO'10], Glocks [IPDPS'11]
  - GPUs: HQL [IPDPS'13]
- Specialized Hardware Support

Expensive communication across NDP units
NDP Synchronization Solution Space

(1) Shared Memory
- Hardware Cache Coherence
- Remote Atomics
- Specialized Hardware Support
  - CPUs: SSB [ISCA'07], Lock Cache [CASES'01], BarrierFilter [MICRO'06]

(2) Message-passing
- Software-based Schemes
  - NDPs: Tesseract [ISCA'15], Near-Data Processing for In-memory Analytics [PACT'15]
  - CPUs: MiSAR [ISCA'15], GPUs: HQL [IPDPS'13]

Lack of a shared level of cache memory
NDP Synchronization Solution Space

(1) Shared Memory
- Hardware Cache Coherence
- Remote Atomics
- Specialized Hardware Support

(2) Message-passing
- Software-based Schemes
- Specialized Hardware Support

Prior schemes are not suitable or efficient for NDP systems
NDP Synchronization Solution Space

(1) Shared Memory
- Hardware Cache Coherence
- Remote Atomics
- Specialized Hardware Support

(2) Message-passing
- Software-based Schemes
- Specialized Hardware Support

SynCron’s Design Choices
- Hardware Message-passing to Avoid Synchronization via Shared Memory
- Hierarchical Communication to Eliminate Expensive Network Traffic
- Specialized Cache Structure to Minimize Latency Costs

NDPs: SynCron [HPCA’21]
Outline

NDP Synchronization Solution Space

Our Mechanism: SyncRon

Evaluation
SynCron: Overview

SynCron consists of four key techniques:

1. **Hardware support** for synchronization acceleration
2. **Direct buffering** of synchronization variables
3. **Hierarchical** message-passing communication
4. Integrated hardware-only **overflow management**
1. Hardware Synchronization Support

- **NDP Unit 0**
  - NDP Core 0
  - NDP Core 1
  - Main Memory
  - Synchronization Engine 0

- **NDP Unit 1**
  - NDP Core 0
  - NDP Core 1
  - Main Memory
  - Synchronization Engine 1

- **ISA**
- **Local lock acquire**

- ✔ No Complex Cache Coherence Protocols
- ✔ No Expensive Atomic Operations
- ✔ Low Hardware Cost
2. Direct Buffering of Variables

NDP Unit 0

NDP Core 0
NDP Core 1
Main Memory
Synchronization Engine 0

Synchronization Processing Unit
Synchronization Table
Indexing Counters

NDP Unit 1

NDP Core 0
NDP Core 1
Main Memory
Synchronization Engine 1

<table>
<thead>
<tr>
<th>Address</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>...</td>
</tr>
<tr>
<td>--</td>
<td>...</td>
</tr>
<tr>
<td>--</td>
<td>...</td>
</tr>
<tr>
<td>--</td>
<td>...</td>
</tr>
</tbody>
</table>

Local lock acquire
2. Direct Buffering of Variables

NDP Unit 0

- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 0

NDP Unit 1

- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 1

- No Costly Memory Accesses
- Low Latency
3. Hierarchical Communication
3. Hierarchical Communication

NPD Unit 0

- NDP Core 0
- NDP Core 1
- Synchronization Engine 0

Main Memory

NPD Unit 1

- NDP Core 0
- NDP Core 1
- Synchronization Engine 1

- syncronVar

NPD Unit 2

- NDP Core 0
- NDP Core 1
- Synchronization Engine 2

Main Memory

NPD Unit 3

- NDP Core 0
- NDP Core 1
- Synchronization Engine 3

Master

Local lock acquire
3. Hierarchical Communication

NDP Unit 0
- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 0

NDP Unit 1
- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 1

NDP Unit 2
- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 2

NDP Unit 3
- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 3

Global lock acquire

Master

SAFARI
3. Hierarchical Communication

- NDP Unit 0
  - NDP Core 0
  - NDP Core 1
  - Main Memory
  - Synchronization Engine 0

- NDP Unit 1
  - NDP Core 0
  - NDP Core 1
  - Main Memory
  - Synchronization Engine 1
  - syncronVar

Minimize Expensive Traffic
4. Integrated Overflow Management

NDP Unit 0

- NDP Core 0
- NDP Core 1
- Main Memory

Synchronization Engine 0

NDP Unit 1

- NDP Core 0
- NDP Core 1
- Main Memory

SyncronVar

Synchronization Engine 1

Synchronization Processing Unit

Synchronization Table

Indexing Counters

Address Table:

<table>
<thead>
<tr>
<th>Address</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x33A9</td>
<td>...</td>
</tr>
<tr>
<td>0x2241</td>
<td>...</td>
</tr>
<tr>
<td>0x438C</td>
<td>...</td>
</tr>
<tr>
<td>0x6B4A</td>
<td>...</td>
</tr>
</tbody>
</table>

syncronVar Address

Counter0 = 0
Counter1 > 0
Counter2 = 0
Counter3 = 0

Master

Fully Occupied
4. Integrated Overflow Management

- Low Performance Degradation
- High Programming Ease
SynCron’s Supported Primitives

**Lock primitive**
- lock_acquire()
- lock_release()

**Barrier primitive**
- barrier_wait_within_NDP_unit()
- barrier_wait_across_NDP_units()

**Semaphore primitive**
- sem_wait()
- sem_post()

**Condition variable primitive**
- cond_wait()
- cond_signal()
- cond_broadcast()
Lock Operation

All NDP cores compete for the same lock variable
Lock Operation

NDP Unit 0

NDP Core 0

NDP Core 1

Main Memory

Synchronization Engine 0

Synchronization Processing Unit

Indexing Counters

Synchronization Table 0

Address | Global Waitlist | Local Waitlist | ...
--- | --- | --- | ---
0x33A9 | 00 | 11 | ...

NDP Unit 1

NDP Core 0

NDP Core 1

Main Memory

Synchronization Engine 1

syncronVar

Master

Synchronization Processing Unit

Indexing Counters

Synchronization Table 1

Address | Global Waitlist | Local Waitlist | ...
--- | --- | --- | ---
0x33A9 | 00 | 11 | ...

SAFARI
Lock Operation

NDP Unit 0

NDP Core 0

NDP Core 1

Main Memory

Synchronization Engine 0

Synchronization Processing Unit

Indexing Counters

Synchronization Table 0

<table>
<thead>
<tr>
<th>Address</th>
<th>Global Waitlist</th>
<th>Local Waitlist</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x33A9</td>
<td>00</td>
<td>11</td>
</tr>
</tbody>
</table>

NDP Unit 1

NDP Core 0

NDP Core 1

Main Memory

Synchronization Engine 1

syncronVar

Synchronization Processing Unit

Indexing Counters

Synchronization Table 1

<table>
<thead>
<tr>
<th>Address</th>
<th>Global Waitlist</th>
<th>Local Waitlist</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x33A9</td>
<td>00</td>
<td>11</td>
</tr>
</tbody>
</table>

Global lock acquire
Lock Operation

NDP Unit 0
- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 0
- Synchronization Processing Unit
- Indexing Counters

Synchro-
- Synchronization Table 0
  - Address: 0x33A9
  - Global Waitlist: 00
  - Local Waitlist: 11

NDP Unit 1
- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 1
- Synchronization Processing Unit
- Indexing Counters

Synchro-
- Synchronization Table 1
  - Address: 0x33A9
  - Global Waitlist: 01
  - Local Waitlist: 11

Global lock acquire

Master
Lock Operation

NDP Unit 0

NDP Core 0
NDP Core 1
Main Memory
Synchronization Engine 0

Synchronization Processing Unit
Indexing Counters

Synchronization Table 0

Address | Global Waitlist | Local Waitlist
---|---|---
0x33A9 | 00 | 11

NDP Unit 1

NDP Core 0
NDP Core 1
Main Memory
Synchronization Engine 1

SyncronVar

Synchronization Processing Unit
Indexing Counters

Synchronization Table 1

Address | Global Waitlist | Local Waitlist
---|---|---
0x33A9 | 01 | 11

Local lock grant

Master

SAFARI
Lock Operation

NDP Unit 0

NDP Core 0
NDP Core 1
Main Memory
Synchronization Engine 0

Synchro-
ization Processing Unit
Indexing Counters

Synchronization Table 0

| Address | Global Waitlist | Local Waitlist | ...
|---------|----------------|---------------|------
| 0x33A9  | 00             | 11            | ...

NDP Unit 1

NDP Core 0
NDP Core 1
Main Memory
Synchronization Engine 1

Global lock grant

Synchronization Table 1

| Address | Global Waitlist | Local Waitlist | ...
|---------|----------------|---------------|------
| 0x33A9  | 01             | 00            | ...
Lock Operation

NDP Unit 0

- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 0

Synchronization Processing Unit

Indexing Counters

Synchronization Table 0

<table>
<thead>
<tr>
<th>Address</th>
<th>Global Waitlist</th>
<th>Local Waitlist</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x33A9</td>
<td>00</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

NDP Unit 1

- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 1
- syncronVar

Synchronization Processing Unit

Indexing Counters

Synchronization Table 1

<table>
<thead>
<tr>
<th>Address</th>
<th>Global Waitlist</th>
<th>Local Waitlist</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x33A9</td>
<td>01</td>
<td>00</td>
</tr>
</tbody>
</table>
Lock Operation

NPD Unit 0

- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 0

Synchronization Processing Unit

Indexing Counters

Synchronization Table 0

<table>
<thead>
<tr>
<th>Address</th>
<th>Global Waitlist</th>
<th>Local Waitlist</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x33A9</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

NPD Unit 1

- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 1

syncronVar

Synchronization Processing Unit

Indexing Counters

Synchronization Table 1

<table>
<thead>
<tr>
<th>Address</th>
<th>Global Waitlist</th>
<th>Local Waitlist</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x33A9</td>
<td>01</td>
<td>00</td>
</tr>
</tbody>
</table>
Lock Operation

NDP Unit 0

- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 0

Synchronization Processing Unit
- Indexing Counters

Synchronization Table 0

| Address | Global Waitlist | Local Waitlist | ...
|---------|----------------|---------------|-----
| --      | --             | --            |     |

NDP Unit 1

- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 1

Synchronization Table 1

| Address | Global Waitlist | Local Waitlist | ...
|---------|----------------|---------------|-----
| 0x33A9  | 01             | 00            |     |

Global lock release

SAFARI
Lock Operation

NDP Unit 0

NDP Core 0

NDP Core 1

Main Memory

Synchronization Engine 0

Synchronization Processing Unit

Indexing Counters

Synchronization Table 0

Address | Global Waitlist | Local Waitlist | ...
---|---|---|---
-- | -- | -- | ...

Master

NDP Unit 1

NDP Core 0

NDP Core 1

Main Memory

Synchronization Engine 1

Synchronization Processing Unit

Indexing Counters

Synchronization Table 1

Address | Global Waitlist | Local Waitlist | ...
---|---|---|---
0x33A9 | 00 | 00 | ...

Global lock release

SAFARI
Lock Operation

NDP Unit 0

- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 0

Synchro-
nization Process-
ing Unit

Indexing Counters

Synchronization Table 0

<table>
<thead>
<tr>
<th>Address</th>
<th>Global Waitlist</th>
<th>Local Waitlist</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
<td>...</td>
</tr>
</tbody>
</table>

NDP Unit 1

- NDP Core 0
- NDP Core 1
- Main Memory
- Synchronization Engine 1

Synchronization Table 1

<table>
<thead>
<tr>
<th>Address</th>
<th>Global Waitlist</th>
<th>Local Waitlist</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
<td>...</td>
</tr>
</tbody>
</table>

Master

Indexing Counters
Lock Operation

More details in the paper
Outline

NDP Synchronization Solution Space

Our Mechanism: SynCron

Evaluation
Evaluation Methodology

• Simulators:
  - **Zsim** [Sanchez+, ISCA’13]
  - **Ramulator** [Kim+, CAL’15]

• System Configuration:
  - 4x NDP units of 16 in-order cores
  - 16KB L1 Data + Instr. Cache
  - 4GB HBM memory

• SynCron’s Default Parameters:
  - Synchronization Processing Unit @1GHz
  - 12-cycle worst-case latency for a message to be served [Aladdin]
  - 64 entries in Synchronization Table, 1-cycle latency [CACTI]
  - 256 entries in indexing counters 2-cycle latency [CACTI]

• Workloads:
  - 9x **Pointer-chasing** Data Structures from ASCYLIB [David+, ASPLOS’15]
  - 6x **Graph Applications** from Crono [Ahmad+, IISWC’15]
  - **Time Series Analysis** from Matrix Profile [Yeh+, ICDM’16]
Comparison Points for SynCron

1. **SynCron**

2. **Central** [Ahn+, ISCA’15]:
   - Synchronization Server: One NDP core of the NDP system
   - Centralized hardware message-passing communication

3. **Hier** [Gao+, PACT’15 / Tang+, ASPLOS’19]:
   - Synchronization Servers: One NDP core per NDP unit
   - Hierarchical hardware message-passing communication

4. **Ideal**
   - Zero overhead for synchronization
Throughput of Pointer Chasing

Stack – 100K

Hash Table – 1K

Linked List – 20K

Central	 Hier	 SynCron	 Ideal

Operations / μs

Operations / ms

Number of NDP Cores

High Contention

Medium Contention

Low Contention

Small # of Variables

Medium # of Variables

High # of Variables

1.18x

1.26x

1.78x

1.68x

1.59x

1.19x
Throughput of Pointer Chasing

SynCron achieves the highest throughput under all scenarios
Speedup in Real Applications

SynCron performs best across all real applications
System Energy in Real Applications

SynCron reduces system energy significantly
### Area and Power Overheads

<table>
<thead>
<tr>
<th></th>
<th>Synchronization Engine</th>
<th>ARM Cortex A7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>40nm</td>
<td>28nm</td>
</tr>
<tr>
<td>Area</td>
<td>9.78%</td>
<td>Total: 0.0461mm²</td>
</tr>
<tr>
<td>Power</td>
<td>2.70%</td>
<td>2.7mW</td>
</tr>
</tbody>
</table>

**SynCron has low area and power overheads**
Sensitivity Studies

- Different memory technologies (HBM, HMC, DDR4)
- Various data placement techniques
- Various transfer latencies on links across NDP units
- Overflow management cost
- Various sizes for the Synchronization Table

**SynCron is effective for a wide variety of configurations**
Summary & Conclusion

• Synchronization is a **major system challenge** for NDP systems

• **Prior** schemes are **not suitable** or **efficient** for NDP systems

• **SynCron** is the **first end-to-end** synchronization solution for NDP architectures

• SynCron consists of **four** key techniques:
  i. **Hardware support** for synchronization acceleration
  ii. **Direct buffering** of synchronization variables
  iii. **Hierarchical** message-passing **communication**
  iv. Integrated hardware-only **overflow management**

• SynCron’s benefits: **90.5%** and **93.8%** of performance and energy of an **Ideal** zero-overhead scheme

• SynCron is **highly-efficient, low-cost, easy-to-use**, and **general** to support many synchronization primitives
SynCron
Efficient Synchronization Support for Near-Data-Processing Architectures

Christina Giannoula
christina.giann@gmail.com

Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas
Ivan Fernandez, Juan Gómez Luna, Lois Orosa
Nectarios Koziris, Georgios Goumas, Onur Mutlu

SAFARI
ETH zürich
Backup Slides
Baseline NDP Architecture

Figure 1: High-level organization of an NDP architecture.
Low Scalability of Coherence-based Synchronization in Real System

<table>
<thead>
<tr>
<th>Million Operations per Second</th>
<th>1 thread single-socket</th>
<th>14 threads single-socket</th>
<th>2 threads same-socket</th>
<th>2 threads different-socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTAS lock [122]</td>
<td>8.92</td>
<td>2.28</td>
<td>9.91</td>
<td>4.32</td>
</tr>
<tr>
<td>Hierarchical Ticket lock [103]</td>
<td>8.06</td>
<td>2.91</td>
<td>9.01</td>
<td>6.79</td>
</tr>
</tbody>
</table>

Table 1: Throughput of two coherence-based lock algorithms on an Intel Xeon Gold server using the libslock library [30].
Low Scalability of Coherence-based Synchronization in NDP Simulated System

Figure 2: Slowdown of a stack data structure using a coherence-based lock over using an ideal zero-cost lock, when varying (a) the NDP cores within a single NDP unit and (b) the number of NDP units while keeping core count constant at 60.
SynCron’s Overview

Figure 3: High-level overview of SynCron.
Lock Operation using SynCron

Figure 4: An example execution scenario for a lock requested by all NDP cores.
SynCron’s API

### SynCron Programming Interface

```c
syncronVar *create_syncvar ();
void destroy_syncvar (syncronVar *svar);
void lock_acquire (syncronVar *lock);
void lock_release (syncronVar *lock);
void barrier_wait_within_unit (syncronVar *bar, int initialCores);
void barrier_wait_across_units (syncronVar *bar, int initialCores);
void sem_wait (syncronVar *sem, int initialResources);
void sem_post (syncronVar *sem);
void cond_wait (syncronVar *cond, syncronVar *lock);
void cond_signal (syncronVar *cond);
void cond_broadcast (syncronVar *cond);
```

**Table 2: SynCron’s Programming Interface (i.e., API).**
Message Encoding of SynCron

![Diagram of message encoding]

**Figure 5:** Message encoding of *SynCron.*
## Message Opcodes for All Primitives

<table>
<thead>
<tr>
<th>Primitives</th>
<th><strong>SynCron</strong> Message Opcodes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Locks</strong></td>
<td>lock_acquire_global, lock_acquire_local, lock_release_global</td>
</tr>
<tr>
<td></td>
<td>lock_release_local, lock_grant_global, lock_grant_local</td>
</tr>
<tr>
<td></td>
<td>lock_acquire_overflow, lock_release_overflow, lock_grant_overflow</td>
</tr>
<tr>
<td><strong>Barriers</strong></td>
<td>barrier_wait_global, barrier_wait_local_within_unit</td>
</tr>
<tr>
<td></td>
<td>barrier_wait_local_across_units, barrier_depart_global, barrier_depart_local</td>
</tr>
<tr>
<td></td>
<td>barrier_wait_overflow, barrier_departure_overflow</td>
</tr>
<tr>
<td><strong>Semaphores</strong></td>
<td>sem_wait_global, sem_wait_local, sem_grant_global</td>
</tr>
<tr>
<td></td>
<td>sem_grant_local, sem_post_global, sem_post_local</td>
</tr>
<tr>
<td></td>
<td>sem_wait_overflow, sem_grant_overflow, sem_post_overflow</td>
</tr>
<tr>
<td><strong>Condition Variables</strong></td>
<td>cond_wait_global, cond_wait_local, cond_signal_global</td>
</tr>
<tr>
<td></td>
<td>cond_signal_local, cond_broad_global, cond_broad_local</td>
</tr>
<tr>
<td></td>
<td>cond_grant_global, cond_grant_local, cond_wait_overflow</td>
</tr>
<tr>
<td></td>
<td>cond_signal_overflow, cond_broad_overflow, cond_grant_overflow</td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td>decrease_indexing_counter</td>
</tr>
</tbody>
</table>

**Table 3: Message opcodes of SynCron.**
The Synchronization Engine

Figure 6: The Synchronization Engine (SE).
Synchronization Table Entry

Figure 7: Synchronization Table (ST) entry.
Control Flow of Message in Synchronization Engine

Figure 8: Control flow in SE.
Synchronization Variable of SynCron

```c
struct syncronVar_t {
    uint16_t Waitlist[4];
    uint64_t VarInfo;
    uint8_t OverflowInfo;
};
typedef struct syncronVar_t syncronVar;
```

**Figure 9: Synchronization variable of SynCron (syncronVar).**
Qualitative Comparison with Closely Related Works

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Primitives</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>ISA Extensions</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Spin-Wait Approach</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Direct Notification</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Target System</td>
<td>uniform</td>
<td>uniform</td>
<td>uniform</td>
<td>non-uniform</td>
</tr>
<tr>
<td>Overflow Management</td>
<td>partially</td>
<td>partially</td>
<td>handled by programmer</td>
<td>fully integrated</td>
</tr>
<tr>
<td></td>
<td>integrated</td>
<td>integrated</td>
<td></td>
<td>integrated</td>
</tr>
</tbody>
</table>

Table 4: Comparison of SynCron with prior mechanisms.
### Simulated System

<table>
<thead>
<tr>
<th>NDP Cores</th>
<th>16 in-order cores @2.5 GHz per NDP unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Data + Inst. Cache</td>
<td>private, 16KB, 2-way, 4-cycle; 64 B line; 23/47 pJ per hit/miss [109]</td>
</tr>
<tr>
<td>NDP Unit Local Network</td>
<td>buffered crossbar network with packet flow control; 1-cycle arbiter; 1-cycle per hop [6]; 0.4 pJ/bit per hop [149]; M/D/1 model [18] for queueing latency;</td>
</tr>
<tr>
<td>DRAM HBM</td>
<td>4 stacks; 4GB HBM 1.0 [92, 93]; 500MHz with 8 channels; nRCDR/nRCDW/nRAS/nWR 7/6/17/8 ns [47, 85]; 7 pJ/bit [151]</td>
</tr>
<tr>
<td>DRAM HMC</td>
<td>4 stacks; 4GB HMC 2.1; 1250MHz; 32 vaults per stack; nRCD/nRAS/nWR 17/34/19 ns [47, 85]</td>
</tr>
<tr>
<td>DRAM DDR4</td>
<td>4 DIMMs; 4GB each DIMM DDR4 2400MHz; nRCD/nRAS/nWR 16/39/18 ns [47, 85]</td>
</tr>
<tr>
<td>Interconnection Links Across NDP Units</td>
<td>12.8GB/s per direction; 40 ns per cache line; 20-cycle; 4 pJ/bit</td>
</tr>
<tr>
<td>Synchronization Engine</td>
<td>SPU @1GHz clock frequency [129]; 8× 64-bit registers; buffer: 280B; ST: 1192B, 64 entries, 1-cycle [109]; indexing counters: 2304B, 256 entries (8 LSB of the address), 2-cycle [109]</td>
</tr>
</tbody>
</table>

Table 5: Configuration of our simulated system.
# Evaluated Workloads

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack [31]</td>
<td>100K - 100% push</td>
</tr>
<tr>
<td>Queue [31, 104]</td>
<td>100K - 100% pop</td>
</tr>
<tr>
<td>Array Map [31, 56]</td>
<td>10 - 100% lookup</td>
</tr>
<tr>
<td>Priority Queue [11, 31, 118]</td>
<td>20K - 100% deleteMin</td>
</tr>
<tr>
<td>Skip List [31, 118]</td>
<td>5K - 100% deletion</td>
</tr>
<tr>
<td>Hash Table [31, 63]</td>
<td>1K - 100% lookup</td>
</tr>
<tr>
<td>Linked List [31, 63]</td>
<td>20K - 100% lookup</td>
</tr>
<tr>
<td>Binary Search Tree Fine-Grained (BST_FG) [130]</td>
<td>20K - 100% lookup</td>
</tr>
<tr>
<td>Binary Search Tree Drachsler (BST_Drachsler) [31, 37]</td>
<td>10K - 100% deletion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Real Application</th>
<th>Locks</th>
<th>Barriers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth First Search (bfs) [7]</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Connected Components (cc) [7]</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Single Source Shortest Paths (sssp) [7]</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Pagerank (pr) [7]</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Teenage Followers (tf) [65]</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Triangle Counting (tc) [7]</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Time Series Analysis (ts) [152]</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Real Application</th>
<th>Input Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfs, cc, sssp,</td>
<td>wikipedia</td>
</tr>
<tr>
<td>pr, tf, tc</td>
<td>-20051105 (wk)</td>
</tr>
<tr>
<td>ts</td>
<td>soc-LiveJournal1 (sl)</td>
</tr>
<tr>
<td></td>
<td>sx-stackoverflow (sx)</td>
</tr>
<tr>
<td></td>
<td>com-Orkut (co)</td>
</tr>
<tr>
<td></td>
<td>air quality (air)</td>
</tr>
<tr>
<td></td>
<td>energy consumption (pow)</td>
</tr>
</tbody>
</table>

**Table 6: Summary of all workloads used in our evaluation.**
Speedup in Simple Microbenchmarks

Figure 10: Speedup of different synchronization primitives.
Throughput of Pointer Chasing

Figure 11: Throughput of pointer chasing using data structures.
Speedup in Real Applications

Figure 12: Speedup in real applications normalized to Central.
Scalability in Real Applications using SynCron

**Figure 13:** Scalability of real applications using *SynCron.*
System Energy in Real Applications

Figure 14: Energy breakdown in real applications for C: Central, H: Hier, SC: SynCron and I: Ideal.
Data Movement in Real Applications

Figure 15: Data movement in real applications for C: Central, H: Hier, SC: SynCron and I: Ideal.
Various Transfer Latencies on Links Across NDP Units – High Contention

Figure 16: Performance sensitivity to the transfer latency of the interconnection links used to connect the NDP units.
Various Transfer Latencies on Links Across NDP Units – Low Contention

Figure 17: Performance sensitivity to the transfer latency of the interconnection links used to connect the NDP units. All data is normalized to *Ideal* (*lower is better*).
Speedup using Different Memory Technologies

Figure 18: Speedup with different memory technologies.
Speedup in Real Applications using two Different Data Placement Techniques

**Figure 19:** Performance sensitivity to a better graph partitioning and maximum ST occupancy of *SynCron.*
Speedup of \textit{Syncron} over its Flat Variant in a Low-contention and Synchronization Non-intensive Scenario

\textbf{Figure 20:} Speedup of \textit{Syncron} normalized to \textit{flat} with 40 ns link latency between NDP units, under a low-contention and synchronization non-intensive scenario.
Speedup of Syncron over its Flat Variant in a Low-contention and Synchronization Intensive Scenario and in a High-contention Scenario

Figure 21: Speedup of SynCron normalized to flat, as we vary the transfer latency of the interconnection links used to connect NDP units, under (a) a low-contention and synchronization-intensive scenario using 4 NDP units, and (b) a high-contention scenario using 2 and 4 NDP units.
## Synchronization Table Occupancy in Real Applications

<table>
<thead>
<tr>
<th>ST Occupancy</th>
<th>Max (%)</th>
<th>Avg (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfs.wk</td>
<td>51</td>
<td>1.33</td>
</tr>
<tr>
<td>bfs.sl</td>
<td>59</td>
<td>1.49</td>
</tr>
<tr>
<td>bfs.sx</td>
<td>51</td>
<td>3.24</td>
</tr>
<tr>
<td>bfs.co</td>
<td>55</td>
<td>6.09</td>
</tr>
<tr>
<td>cc.wk</td>
<td>63</td>
<td>1.27</td>
</tr>
<tr>
<td>cc.sl</td>
<td>61</td>
<td>2.16</td>
</tr>
<tr>
<td>cc.sx</td>
<td>48</td>
<td>2.43</td>
</tr>
<tr>
<td>cc.co</td>
<td>46</td>
<td>4.53</td>
</tr>
<tr>
<td>sssp.wk</td>
<td>62</td>
<td>1.18</td>
</tr>
<tr>
<td>sssp.sl</td>
<td>54</td>
<td>2.08</td>
</tr>
<tr>
<td>sssp.sx</td>
<td>50</td>
<td>2.20</td>
</tr>
<tr>
<td>sssp.co</td>
<td>48</td>
<td>5.23</td>
</tr>
<tr>
<td>pr.wk</td>
<td>62</td>
<td>4.27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ST Occupancy</th>
<th>Max (%)</th>
<th>Avg (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr.sl</td>
<td>51</td>
<td>2.27</td>
</tr>
<tr>
<td>pr.sx</td>
<td>53</td>
<td>2.46</td>
</tr>
<tr>
<td>pr.co</td>
<td>48</td>
<td>4.72</td>
</tr>
<tr>
<td>tf.wk</td>
<td>62</td>
<td>1.44</td>
</tr>
<tr>
<td>tf.sl</td>
<td>53</td>
<td>2.21</td>
</tr>
<tr>
<td>tf.sx</td>
<td>50</td>
<td>2.99</td>
</tr>
<tr>
<td>tf.co</td>
<td>48</td>
<td>4.61</td>
</tr>
<tr>
<td>tc.wk</td>
<td>62</td>
<td>1.26</td>
</tr>
<tr>
<td>tc.sl</td>
<td>48</td>
<td>2.08</td>
</tr>
<tr>
<td>tc.sx</td>
<td>50</td>
<td>2.77</td>
</tr>
<tr>
<td>tc.co</td>
<td>51</td>
<td>4.52</td>
</tr>
<tr>
<td>ts.air</td>
<td>84</td>
<td>44.20</td>
</tr>
<tr>
<td>ts.pow</td>
<td>89</td>
<td>43.51</td>
</tr>
</tbody>
</table>

**Table 7: ST occupancy in real applications.**
Slowdown in Real Applications when Varying the Synchronization Table Size

Figure 22: Slowdown with varying ST size (normalized to 64-entry ST). Numbers on top of bars show the percentage of overflowed requests.
Overflow Management Cost of Different Overflow Schemes

**Figure 23:** Throughput achieved by BST_FG using different overflow schemes and varying the ST size. The reported numbers show the percentage of overflowed requests.
Area and Power Overheads of the Synchronization Engine

<table>
<thead>
<tr>
<th></th>
<th>SE (Synchronization Engine)</th>
<th>ARM Cortex A7 [14]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Technology</strong></td>
<td>40nm</td>
<td>28nm</td>
</tr>
<tr>
<td><strong>Area</strong></td>
<td>SPU: 0.0141mm², ST: 0.0112mm²</td>
<td>32KB L1 Cache</td>
</tr>
<tr>
<td></td>
<td>Indexing Counters: 0.0208mm²</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Total:</strong> 0.0461mm²</td>
<td><strong>Total:</strong> 0.45mm²</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>2.7 mW</td>
<td>100mW</td>
</tr>
</tbody>
</table>

Table 8: Comparison of SE with a simple general-purpose in-order core, ARM Cortex A7.