HASHI: An Application-Specific Instruction Set Extension for Hashing

Oliver Arnold, Sebastian Haas, Gerhard Fettweis,
Benjamin Schlegel, Thomas Kissinger,
Tomas Karnagel, Wolfgang Lehner

Technische Universität Dresden
Dresden, Germany

Motivation

TECHNISCHE UNIVERSITÄT DRESDEN

Today's Database Systems

- Fat Cores (area & power)
- Few HW adaptions
- CMOS Scaling

Database Processors

- Processors build from scratch
- Long development cycle
- High development costs

Our Approach

- HW/SW codesign
- Customizable processor
- Hashing-specific ISA extensions
- Tool flow ➔ short HW development cycles

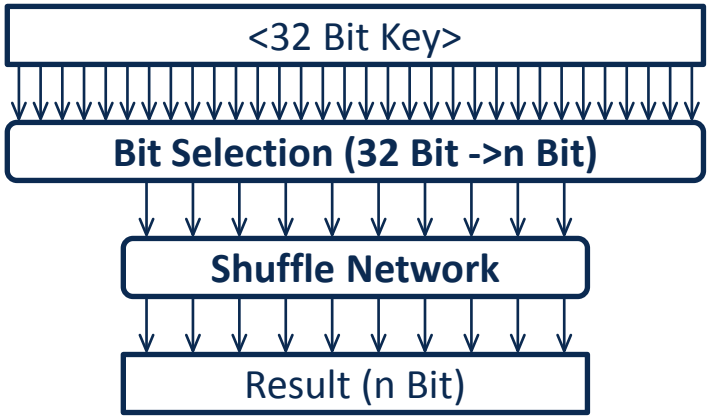
Application Scenario 1: Integer Hash Function

- Bit Extraction

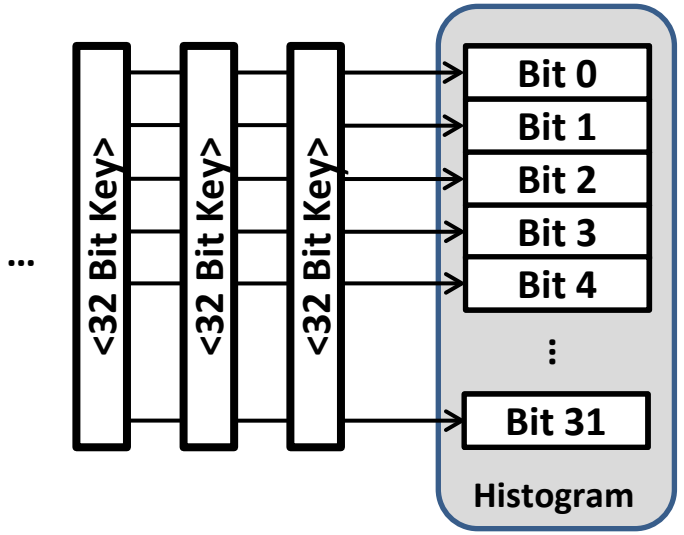
  Selection of specific bits in a 32-bit key via arbitrary hash mask

- Sampling

  Scanning a subset of the data set to choose the most efficient hash mask



Bit Extraction



Sampling

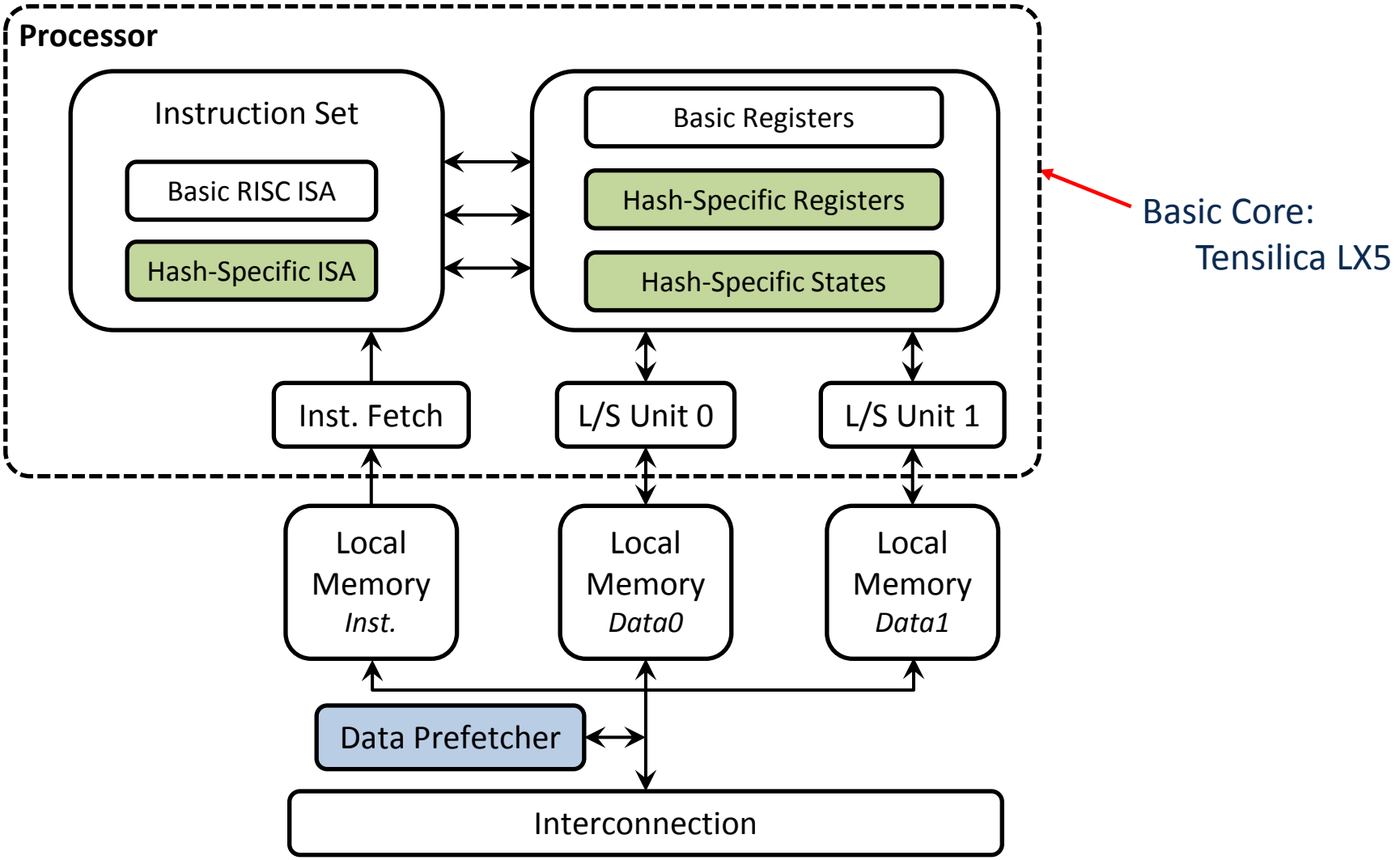
Application Scenario 2

- CityHash32
  - Non-cryptographic hash function for strings
  - Returns 32-bit hash value

```
unsigned int CityHash32(char *s, int len){
      int hash = comp_1(s+len-20);
      int i = (len-1)/20;

      do {
            hash = comp_2(s, hash);
            s += 20;
      } while(--i != 0);

      return comp_3(hash);
}
```

- Hash Table Operators (Insert, Lookup)
  - Operate on 32-bit keys
  - Apply integer hash function

Customizable Processor Model



Integer Hash Function: C code

```
unsigned int hash, shVal, shVal_neg;
unsigned int mask = 0xFFFFFFFF;

for(i=0; i<keySize; i++){
    //load key, bit selection
    hash = key[i] & hashFunc;

    //extract bits
    for(j=30; j>=0; j--){
        if(!(hashFunc & (0x1<<j))){
            //partial shift right
            shVal = hash & (mask<<j);
            shVal_neg = hash & ~(mask<<j);
            hash = (shVal>>1) | shVal_neg;
        }
    }
    //store hash value
    hashValue[i] = hash;
}
```

Pure C code

Integer Hash Function: C code

```
unsigned int hash, shVal, shVal_neg;
unsigned int mask = 0xFFFFFFFF;

for(i=0; i<keySize; i++){
    //load key, bit selection
    hash = key[i] & hashFunc;

    //extract bits
    for(j=30; j>=0; j--){
        if(!(hashFunc & (0x1<<j))){
            //partial shift right
            shVal = hash & (mask<<j);
            shVal_neg = hash & ~(mask<<j);
            hash = (shVal>>1) | shVal_neg;
        }
    }
    //store hash value
    hashValue[i] = hash;
}
```

Pure C code

```
//init pointer, variables
init_states(key, hashValue, hashFunc);

LD_0(); LD_1();

//load keys, extract bits, store hash values
for(i=0; i<(keySize/16); i++){
    LD_0(); LD_1(); HOP();
    LD_0(); LD_1();
    HOP(); ST_0(); ST_1();
}

HOP();
ST_0(); ST_1();
```

C code with new instructions

Integer Hash Function: C code

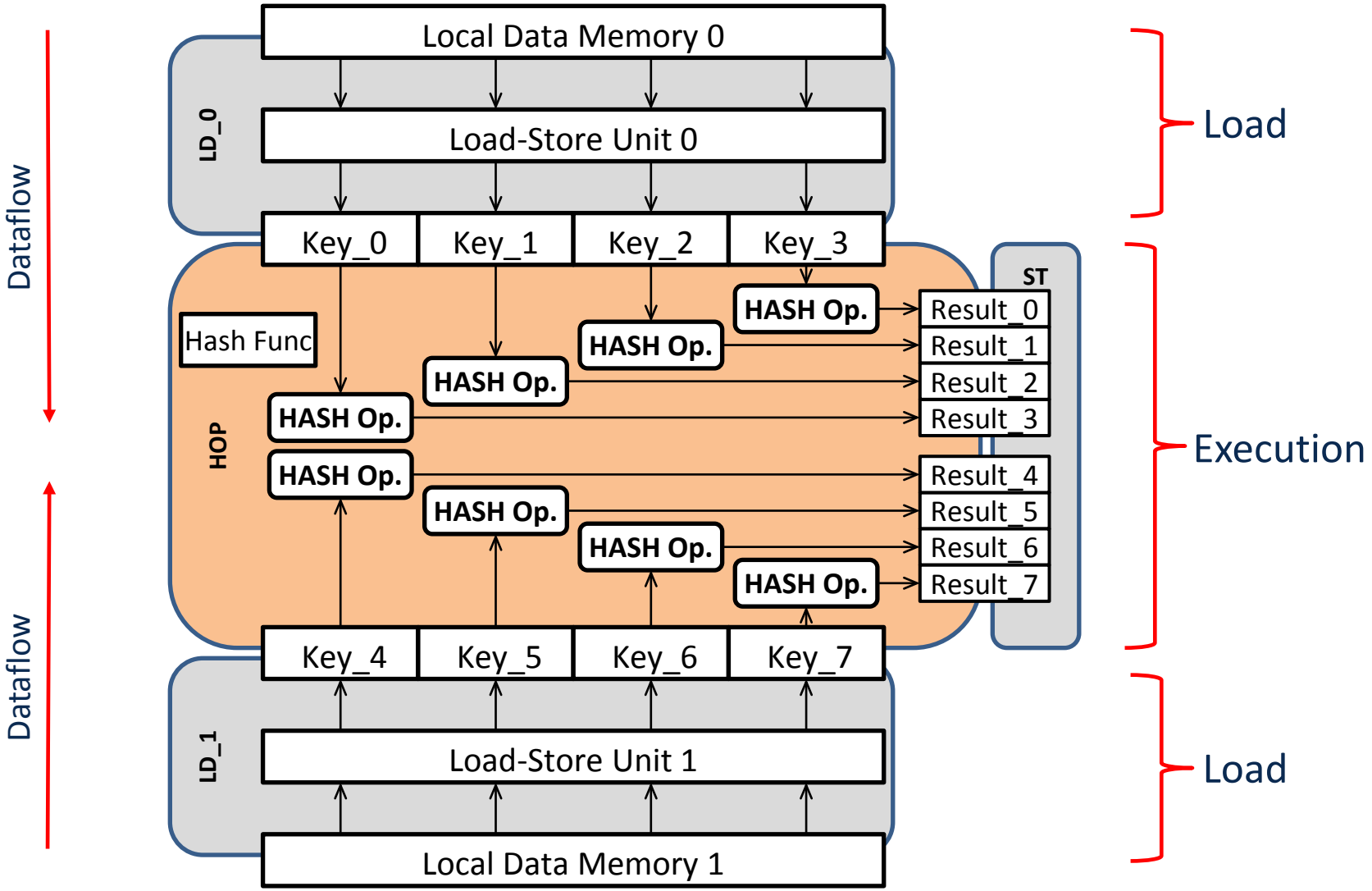
```
unsigned int hash, shVal, shVal_neg;
unsigned int mask = 0xFFFFFFFF;

for(i=0; i<keySize; i++){
    //load key, bit selection
    hash = key[i] & hashFunc;

    //extract bits
    for(j=30; j>=0; j--){
        if(!(hashFunc & (0x1<<j))){
            //partial shift right
            shVal = hash & (mask<<j);
            shVal_neg = hash & ~(mask<<j);
            hash = (shVal>>1) | shVal_neg;
        }
    }
    //store hash value
    hashValue[i] = hash;
}
```

Pure C code

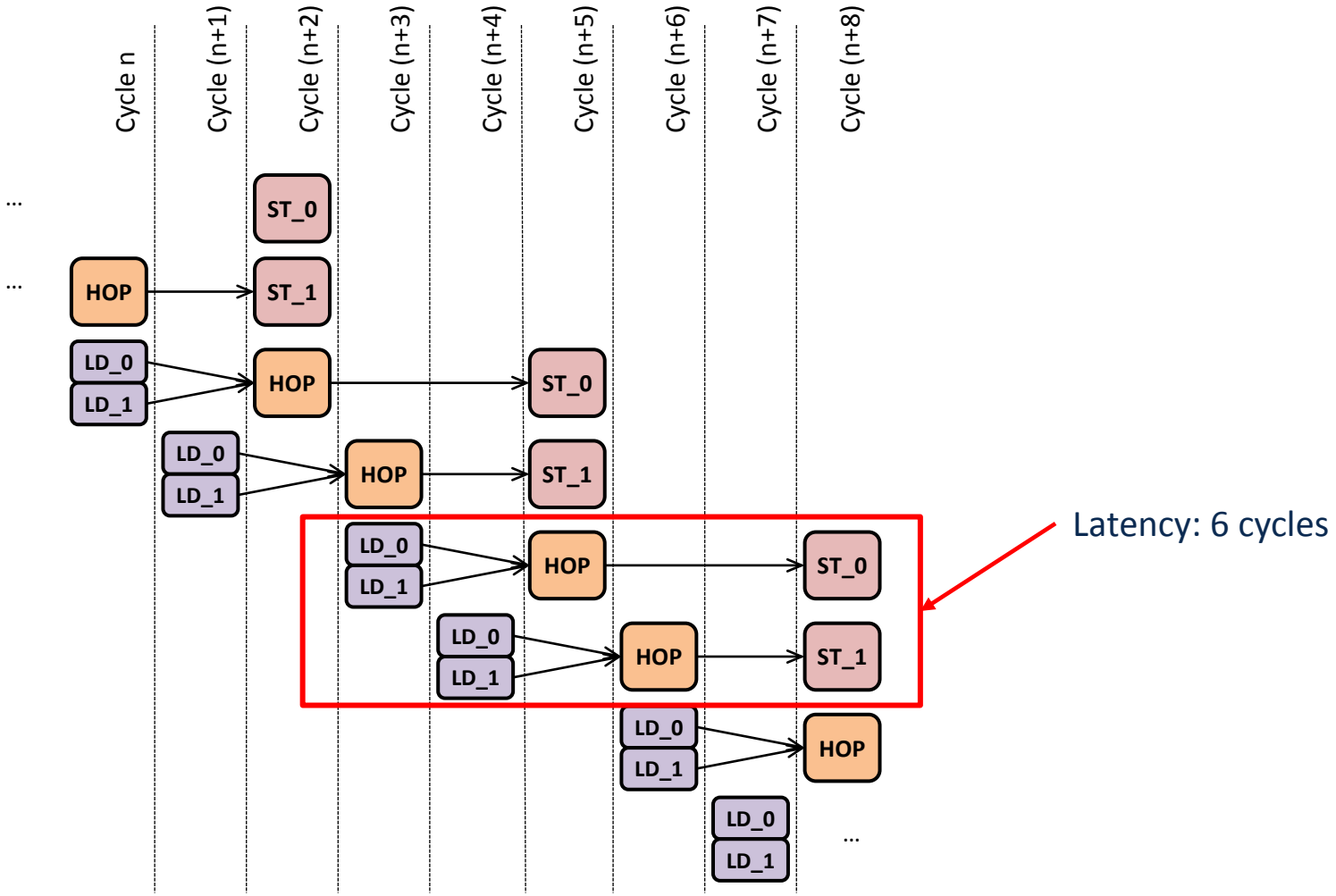
```
//init pointer, variables
init_states(key, hashValue, hashFunc);

LD_0(); LD_1();

//load keys, extract bits, store hash values
for(i=0; i<(keySize/16); i++){
    LD_0(); LD_1(); HOP();          1 cycle
    LD_0(); LD_1();                 1 cycle
    HOP(); ST_0(); ST_1();          1 cycle
}

HOP();
ST_0(); ST_1();
```

C code with new instructions

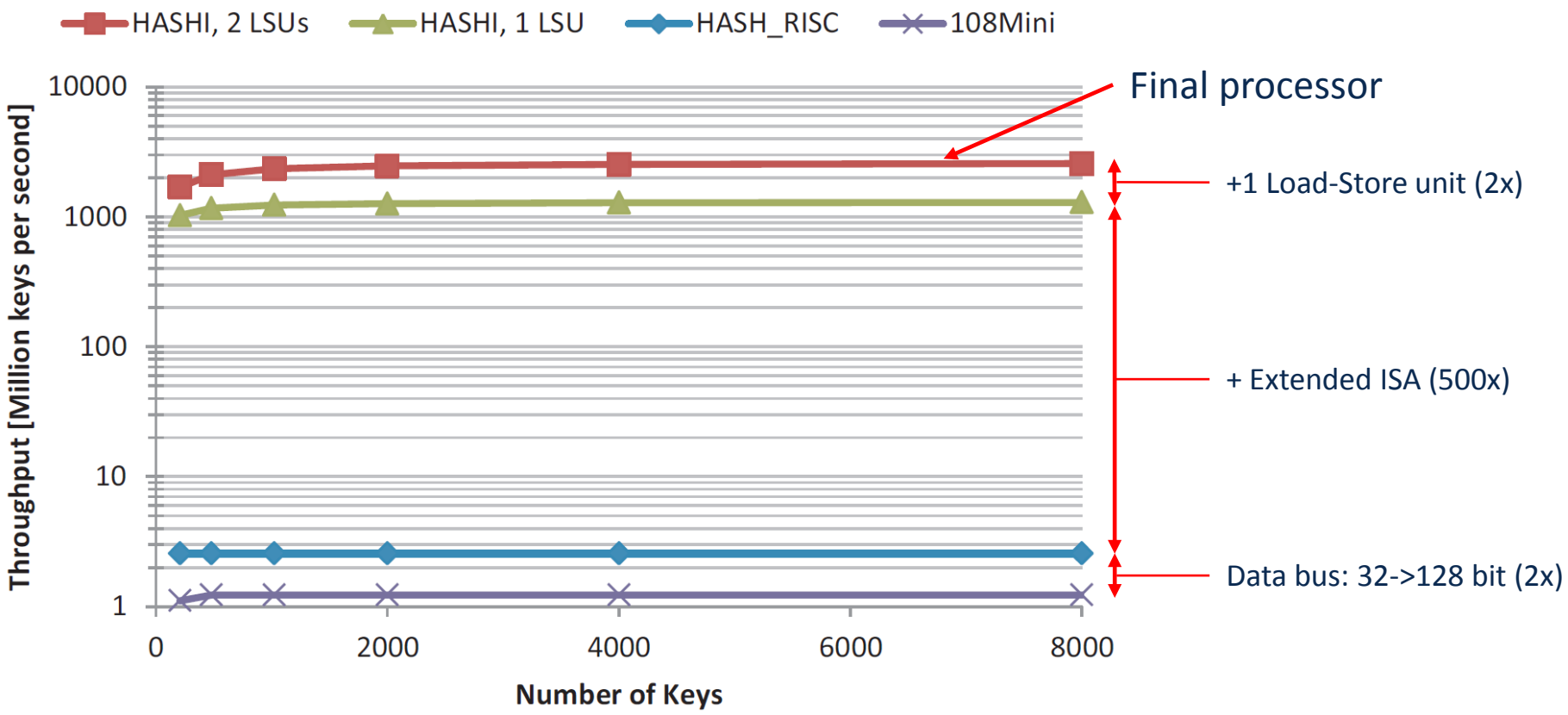
Integer Hash Function: ISA Extensions



Integer Hash Function: Pipeline Snippet



Integer Hash Function: Throughput



Throughput $T = \frac{n_{key}}{t}$

n_{key}: number of keys

t: time to perform the operation

Results: Throughput

Benchmark	108Mini	HASH_RISC	HASHI
Frequency [MHz]	442	555	488
Hash + Lookup [MHashs/s]	1.0	2.1	386
Hash + Insert [MHashs/s]	1.1	2.3	389
Hash Keys [MHashs/s]	1.1	2.4	2,533
Hash Sampling [MHashs/s]	2.0	3.4	2,575
CityHash32 [MChars/s]	38.3	64.5	4,770

Final processor

Speedup:
HASHI *vs.* 108Mini

→ **386x**
→ **354x**
→ **2303x**
→ **1288x**
→ **125x**

Results: Timing and Area

Process	Processor	A_{LOGIC} [mm^2]	A_{MEM} [mm^2]	f_{MAX} [MHz]	P[mW] @f_{MAX}
65 nm	108Mini	0.220[1]	-	442[1]	27.4[1]
	HASH_RISC	0.164	0.874	555	63.2
	HASHI	0.731	0.874	488	138.4
28 nm	HASHI	0.214	0.213	500	-

[1] http://ip.cadence.com/uploads/pdf/108Mini.pdf

Final processor

Relative Area Consumption (HASHI)

Part	Area[%]
Basic Core	18.4
Hash + Lookup	4.4
Hash + Insert	26.9
Hash Keys	5.7
Hash Sampling	13.3
CityHash32	25.8
ALL	5.5
SUM	100

Results: Comparison

	HASHI	Intel i7-4550U (Haswell)	Intel i7-3960X (Sandy-Bridge)
Technology [nm]	65	22	32
Frequency [GHz]	0.488	1.5 (3.0[3])	3.3 (3.9[3])
Power Consumption [W]	0.138	7.9[1]	24.3[1]
Area [mm^2]	1.605	181	434.7
Hash Keys [MHashs/s]	2,533	20.6 2,063[2]	14.9
Sampling [MHashs/s]	2,575	14.5	14.0
CityHash32 [MChars/s]	4,770	4,720	3,678

[1]Measured with RAPL Counter (only core power: PP0)
[2]w/ Intel AVX2-PEXT instruction
[3]Max turbo frequency

Measures:
HASHI *vs.* Intel

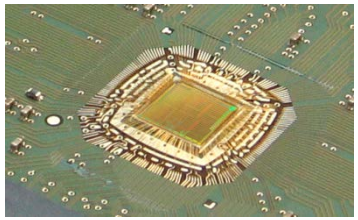
→ **3x/7x lower**

→ **57x/176x lower**

→ **113x/271x lower**

Conclusion

- Hardware/Software Codesign approach
- Results
  - High database throughput
  - Highly reduced area and power consumption
  - 170x less energy consumption than a high-end x86 processor (@ same performance)
- Silicon Prototype
  - Tape-out April 2014
  - 28 nm LP process: Globalfoundries
  - ISA: Hash Functions, Hash Table Operators etc.



[1]

1 Nöthen et al., A 105GOPS 36mm2 Heterogeneous SDR MPSoC with Energy-Aware Dynamic Scheduling and Iterative Detection-Decoding for 4G in 65nm CMOS, ISSCC. 2014