Some Useful Microarchitectural Techniques for 128-bit Support in General Purpose Processors

Arthur Perais (arthur.perais@univ-grenoble-alpes.fr), CNRS*

Journée thématique du club des partenaires sur le RISC-V - 29 Septembre 2023

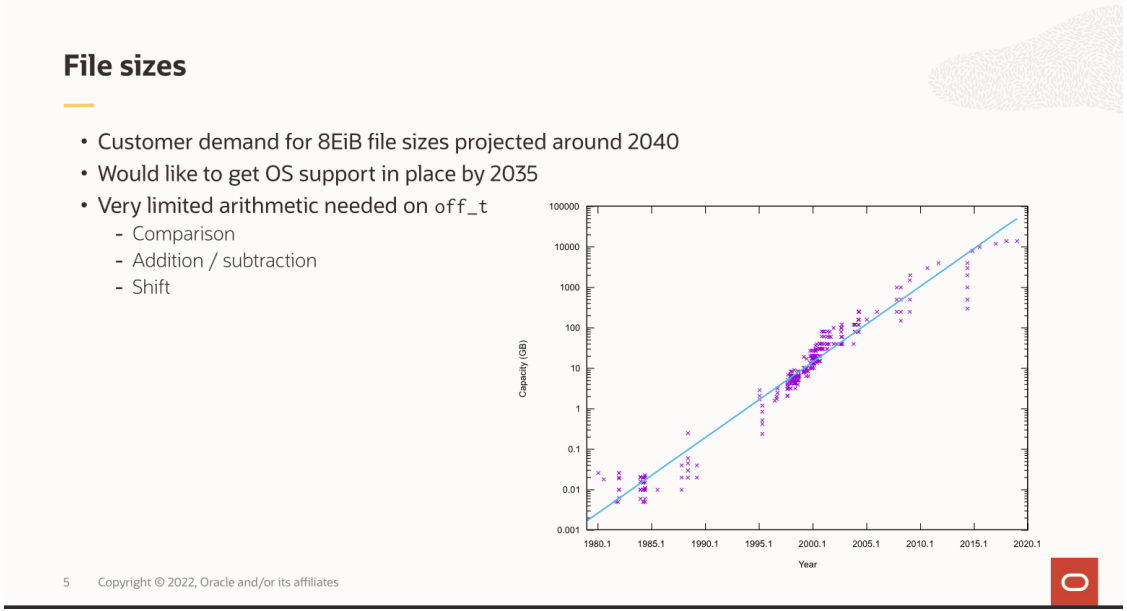
*Laboratoire TIMA @ CNRS, UGA, Grenoble INP

This Presentation…

- **…will not convince you 128-bit machines are around the corner**
- **…should be seen as « what if 128-bit » from the CPU microarchitecture standpoint**
- **…some ideas could be « backported » to 64-bit microarchitectures**
  - From Deshpande, Perais and Pétrot in IEEE Computer Architecture Letters

128-bit Machines

- **A vague notion that such machines could be needed in some years**
  - Larger and larger files, log2(8 EiB) = 63



From “Zettalinux: It's Not Too Late To Start”, Matthew Wilcox, Linux Plumbers Conference 2022

128-bit Machines

- **A vague notion that such machines could be needed in some years**
  - Larger and larger files, log2(8 EiB) = 63
  - Naively following the trend, we will get there

- **Just thinking about what this implies for CPU uarch**
  - Assume RISC-V vision of « as usual » 128-bit: RV128
  - Don’t think about virtual memory, what the system looks like
    - Others are looking at it (ANR project Maplurinum)

128-bit CPU Microarchitecture

- **What do I need in my design to execute RV-128 ?**
  - 128-bit physical registers

128-bit CPU Microarchitecture

- **What do I need in my design to execute RV-128 ?**
  - 128-bit physical registers
  - 128-bit operators

128-bit CPU Microarchitecture

- **What do I need in my design to execute RV-128 ?**
  - 128-bit physical registers
  - 128-bit operators
  - 128-bit datapaths
    - regs to operators
    - operators to regs
    - operators to operators (bypass)

128-bit CPU Microarchitecture

- **What do I need in my design to execute RV-128 ?**
  - 128-bit physical registers
  - 128-bit operators
  - 128-bit datapaths
    - regs to operators
    - operators to regs
    - operators to operators (bypass)
  - Larger tags in caches, BTB, TLBs
    - Probably ok at first as will implement 65-bit VA, not 128-bit VA
    - Still, tags cost more and more as implemented VA/PA grows to architectural size

128-bit CPU Microarchitecture

- **What do I need in my design to execute RV-128 ?**
  - 128-bit physical registers
  - 128-bit operators
  - 128-bit datapaths
    - regs to operators
    - operators to regs
    - operators to operators (bypass)
  - Larger tags in caches, BTB, TLBs
    - Probably ok at first as will implement 65-bit VA, not 128-bit VA
    - Still, tags cost more and more as implemented VA/PA grows to architectural size
  - ➢ Costs area, hence latency and power: « HW » tax

128-bit CPU Microarchitecture

- **What do I need in my design to execute RV-128 ?**
  - 128-bit physical registers
  - 128-bit operators
  - 128-bit datapaths
    - regs to operators
    - operators to regs
    - operators to operators (bypass)
  - Larger tags in caches, BTB, TLBs
    - Probably ok at first as will implement 65-bit VA, not 128-bit VA
    - Still, tags cost more and more as implemented VA/PA grows to architectural size
  - ➢ Costs area, hence latency and power: « HW » tax

- **And, pointers occupy 128-bit in caches**
  - ➢ Less effective caching (4 vs. 8 pointers per 64B cacheline): « SW » tax

128-bit CPU Microarchitecture

- **HW + SW tax**
  - 128-bit code will run slower
- **Let’s try to do something about it**
  - Not obvious what to do about 128-bit pointers and caching
  - But we can do something about hardware overheads in the CPU
    - Maybe we can apply this « something » to 64-bit CPUs

Compiling Code for a 128-bit Machine

```
int main()
{
  for(int i = 0; i < 64; i++)
  {
    c[i] = a[i] + b[i] * 10;
  }
}
```

gcc 13 –O1 →

```
loop:
  lw      a1,0(a3)    // *b
  slliw   a5,a1,2     // *b * 4
  addw    a5,a5,a1    // *b * 5
  slliw   a5,a5,1     // *b * 10
  lw      a1,0(a4)    // *a
  addw    a5,a5,a1    // *a + *b * 10
  sw      a5,0(a2)    // *c = ...
  addi    a4,a4,4     // a++
  addi    a3,a3,4     // b++
  addi    a2,a2,4     // c++
  addi    a6,a6,1     // i++
  bne     a6,a0, loop
```

Compiling Code for a 128-bit Machine

```
int main()
{
  for(int i = 0; i < 64; i++)
  {
    c[i] = a[i] + b[i] * 10;
  }
}
```

gcc 13 –O1 →

```
loop:
  lw      a1,0(a3)    // *b
  slliw   a5,a1,2     // *b * 4
  addw    a5,a5,a1    // *b * 5
  slliw   a5,a5,1     // *b * 10
  lw      a1,0(a4)    // *a
  addw    a5,a5,a1    // *a + *b * 10
  sw      a5,0(a2)    // *c = ...
  addi    a4,a4,4     // a++
  addi    a3,a3,4     // b++
  addi    a2,a2,4     // c++
  addi    a6,a6,1     // i++
  bne     a6,a0, loop
```

What actually needs to use 128-bit ?

Compiling Code for a 128-bit Machine

```
int main()
{
  for(int i = 0; i < 64; i++)
  {
    c[i] = a[i] + b[i] * 10;
  }
}
```

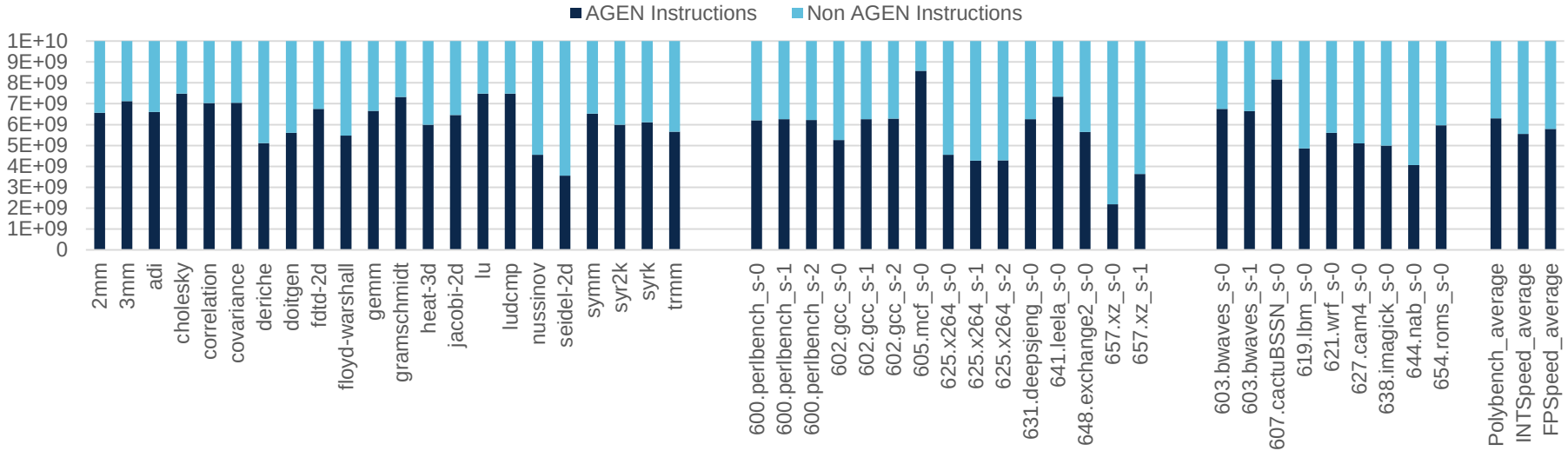
gcc 13 –O1 →

```
loop:
  lw    a1,0(a3)    // *b
  slliw  a5,a1,2    // *b * 4
  addw  a5,a5,a1    // *b * 5
  slliw  a5,a5,1    // *b * 10
  lw    a1,0(a4)    // *a
  addw  a5,a5,a1    // *a + *b * 10
  sw    a5,0(a2)    // *c = ...
  addi  a4,a4,4     // a++
  addi  a3,a3,4     // b++
  addi  a2,a2,4     // c++
  addi  a6,a6,1     // i++
  bne   a6,a0, loop
```

Address generation (in blue) !

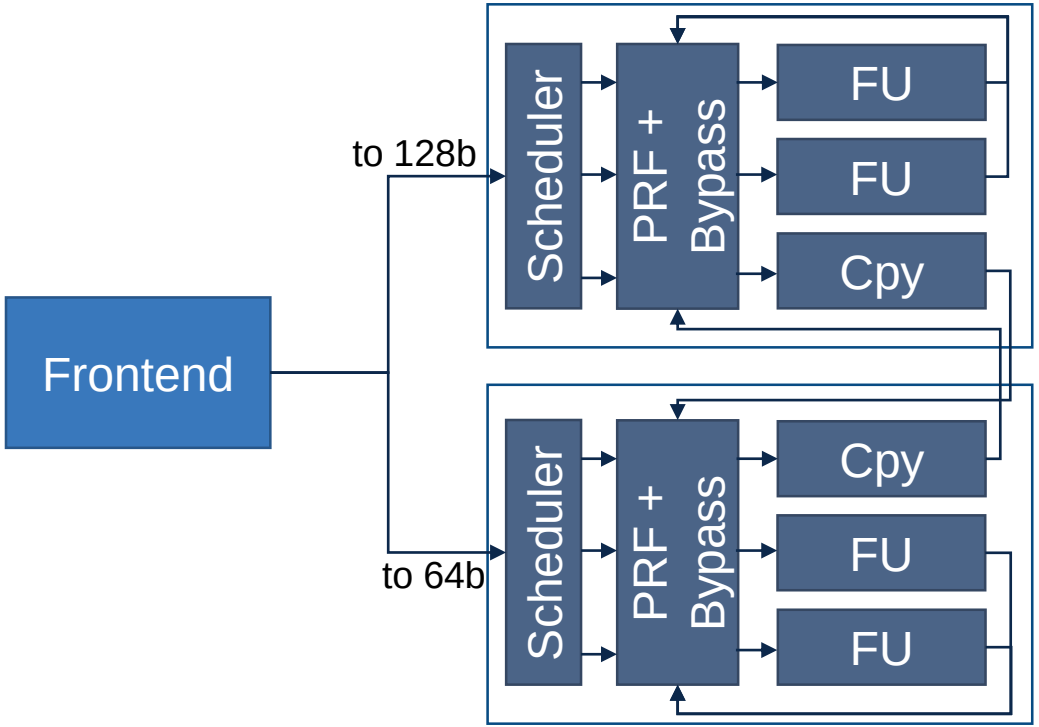
Percentage of Instructions Doing AGEN

- **Polybench (10B instructions) /SPEC2k17 (100B warmup, 10B instructions) ran on Spike**
  - Count how many instructions participate to AGEN, those would manipulate 128-bit data
    - Load/Store/Indirect branches
    - Their producers



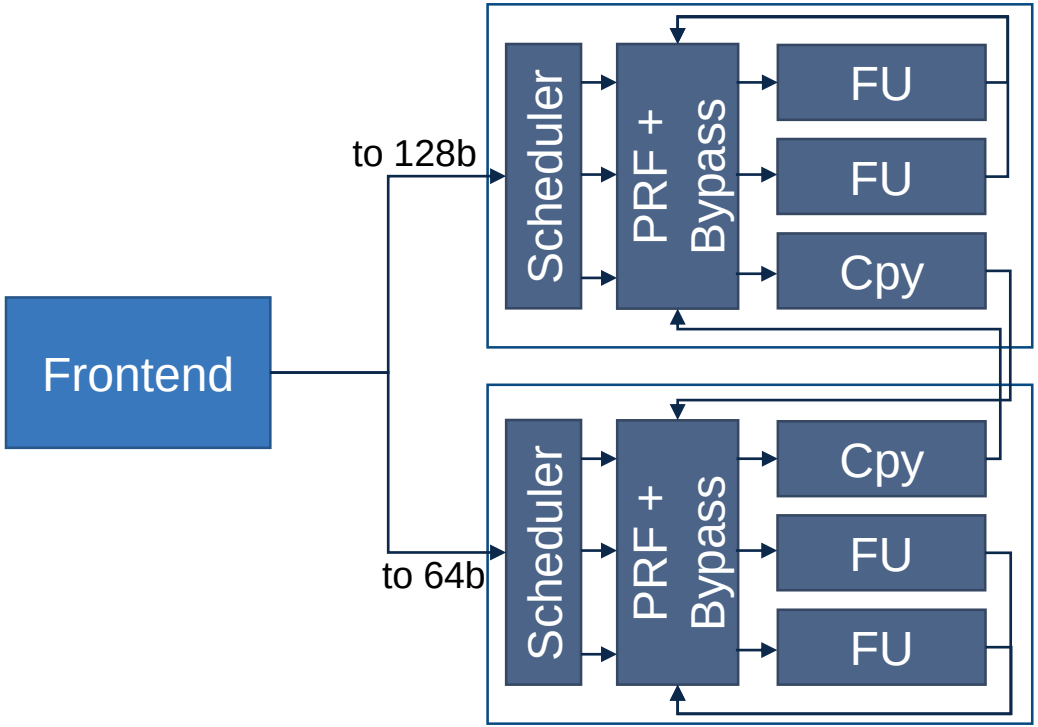
Address/Value Clustered Microarchitecture

- **Block a.k.a. cluster, each having own**
  - Physical register file
  - Scheduler
  - Functional units
  - Dense « local » bypass
  - On-demand « global » bypass



Address/Value Clustered Microarchitecture

- **Block a.k.a. cluster, each having own**
  - Physical register file
  - Scheduler
  - Functional units
  - Dense « local » bypass
  - On-demand « global » bypass
- **Frontend *steers* instructions to clusters**
  - In baseline clustering, trying to keep occupancy balanced
  - In our proposal, based on 128-bit requirement



Clustering Pros & Cons

- **Divide & Conquer approach**
  - If code well balanced, same throughput as a monolithic cluster
  - Lower complexity than monolithic cluster as some structures do not scale linearly (latency)

- **Heterogeneous clustering**
  - Only implement 128-bit FUs used for AGEN
    - Not many division on address generation slices
      - ➢ Microcoded or in software

Clustering Pros & Cons

- **Drawbacks**
  - If code not well balanced, lower throughput than monolithic cluster
  - Need to identify 32/64-bit instructions that should go to 128-bit cluster
    - **Speculative** steering
  - When incorrect steering, penalty to explicitly copy value from one cluster to another

```
movi   a3, 0x42     // b =>steer to INT cluster
…
lw     a1,0(a3)     // *b => steer to AGEN cluster, copy a3 from INT cluster (costs 1 inst)
```

128-bit CPU Microarchitecture

- **Clustering limits complexity increase in the backend**
  - Technique could be envisioned for 64-bit machines
    - 64-bit AGEN cluster, 64-bit INT cluster
    - Basically a new twist on « well-known » clustering idea
      - ✓ Disjoint register files
      - ✓ Data type driven steering

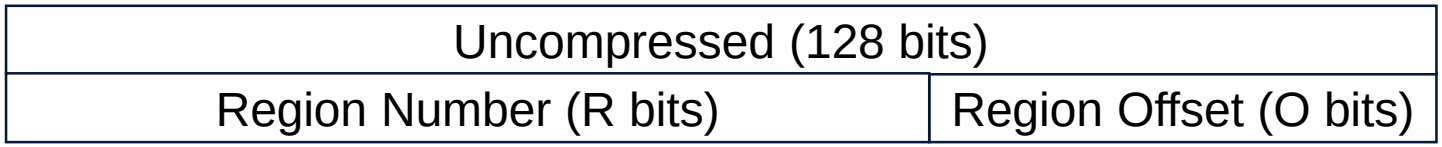
- **We still need large structures in the AGEN cluster**
  - Many 128-bit physical registers to support a large instruction window

Compressed Physical Register (PRF)

- **Baseline INT PRF contains both addresses and general purpose values**
- **We expect AGEN cluster PRF only contains addresses**
  - Addresses have high locality (why caches work) !
- **Use region-based compression to greatly reduce register size**

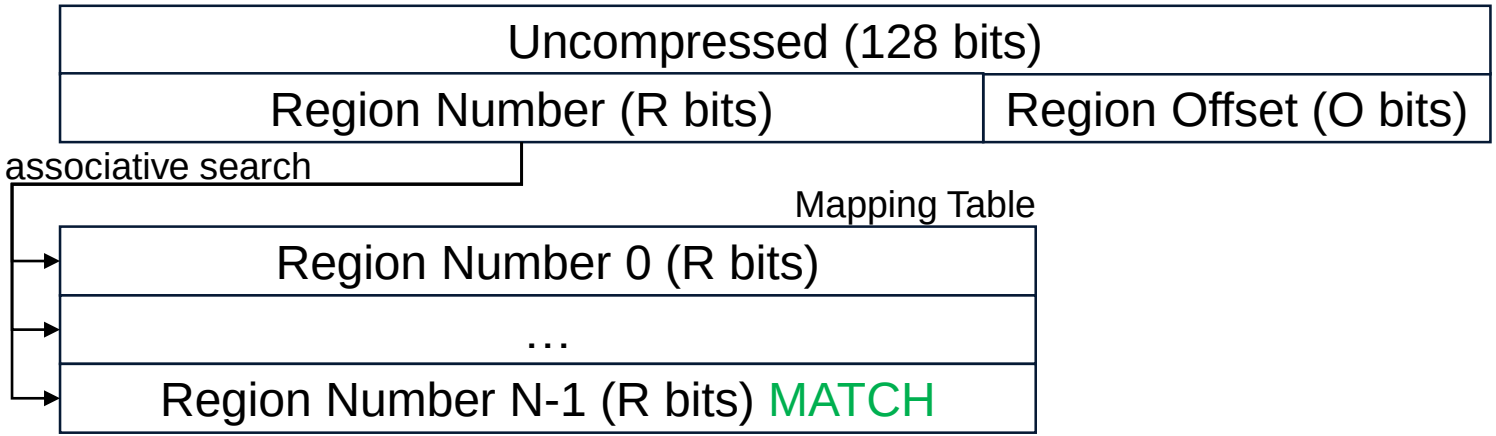
Region-based Compression

- **Represent a value as (region number, region offset)**



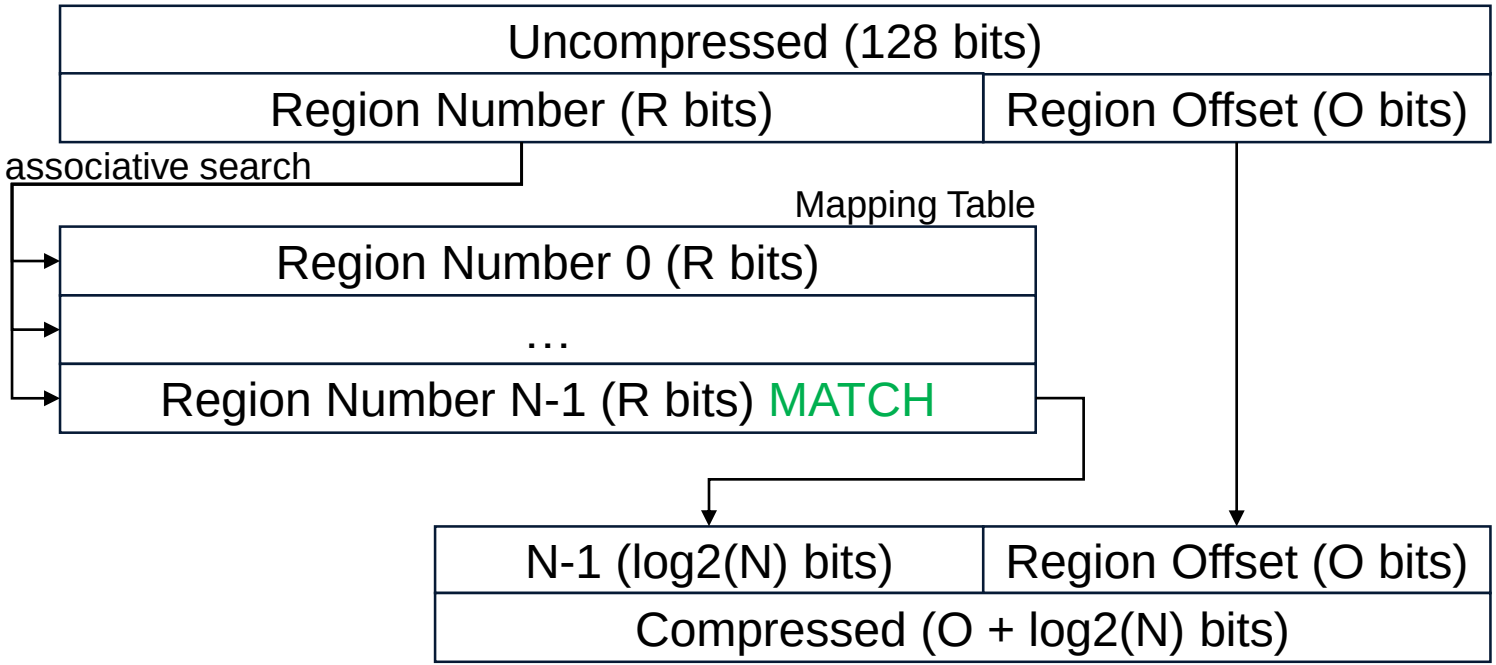
Region-based Compression

- **Compression (e.g., at FU output to write to PRF)**



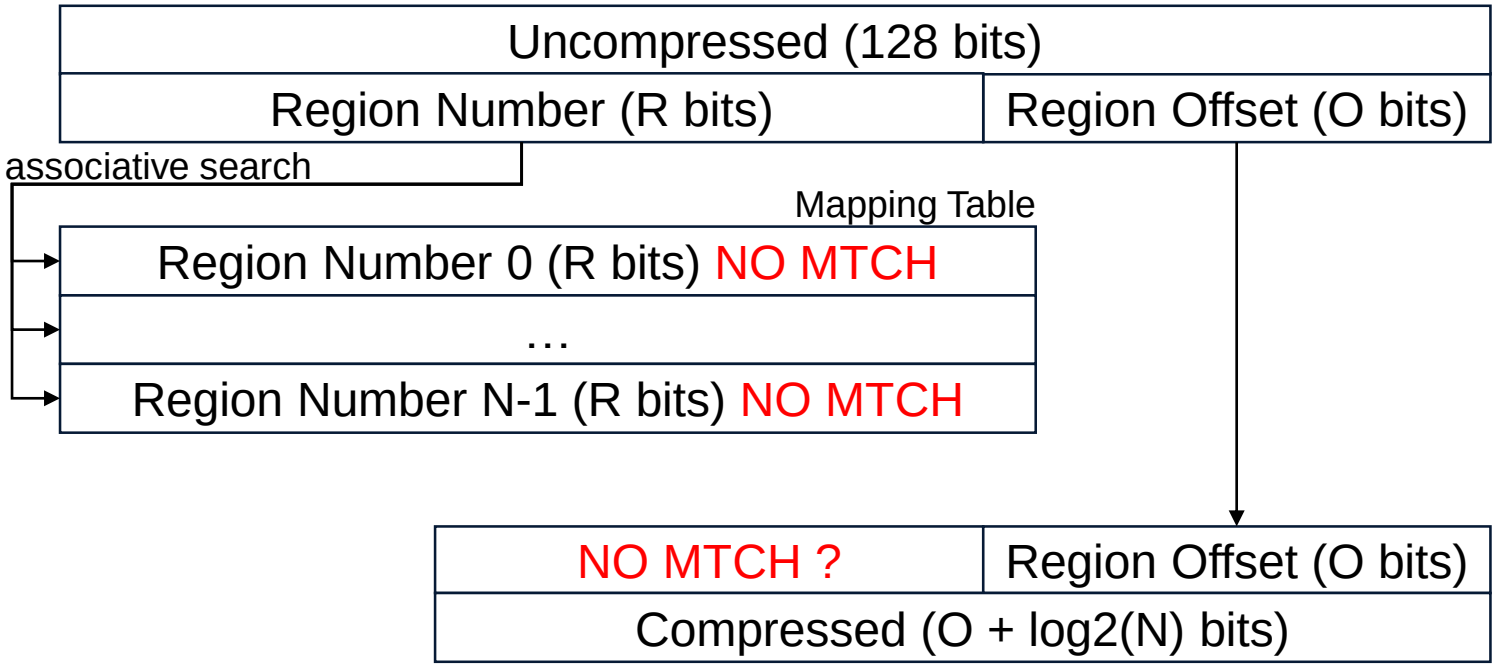
Region-based Compression

- **Compression (e.g., at FU output to write to PRF)**



Region-based Compression

- **Compression (e.g., at FU output to write to PRF)**

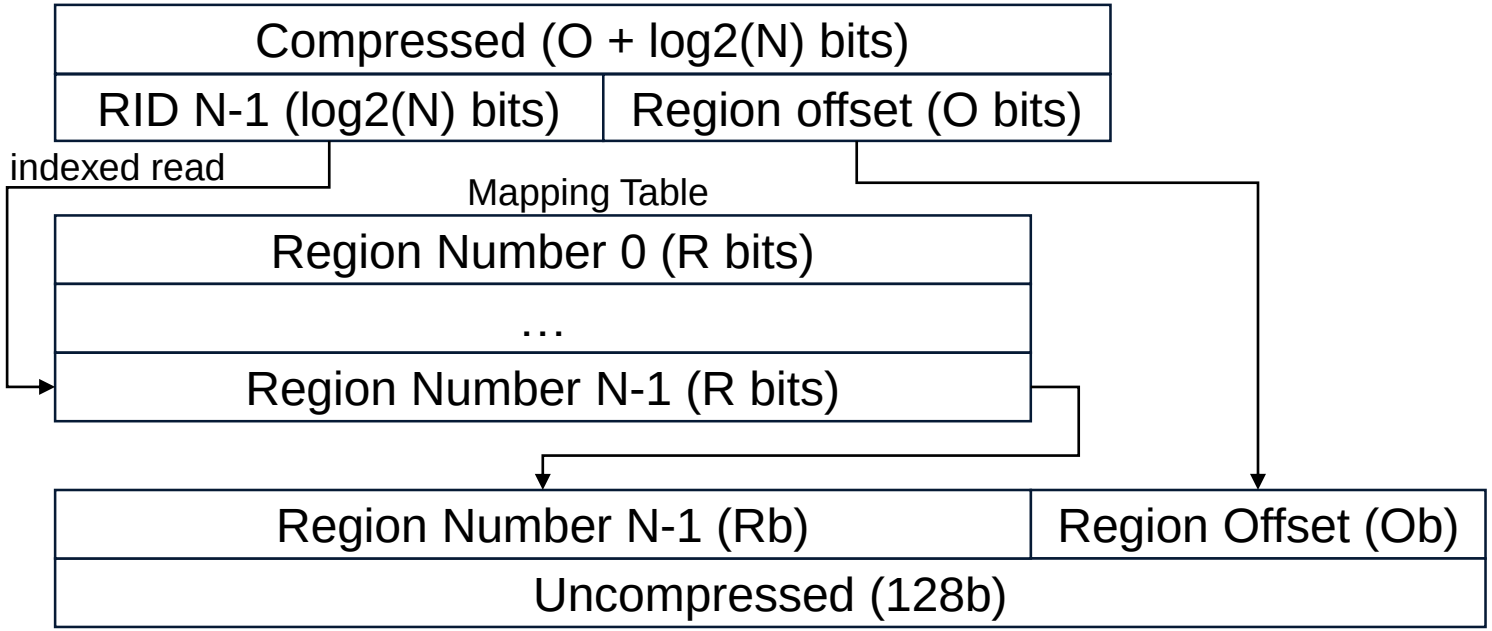


Region-based Compression

- **Compression (e.g., at FU output to write to PRF)**
- **Mismatch in region table requires reclaiming a region**
  - Not going into too much details
    - Fine if have at least 32 entries in the table
    - Will be slow (O(#pregs))

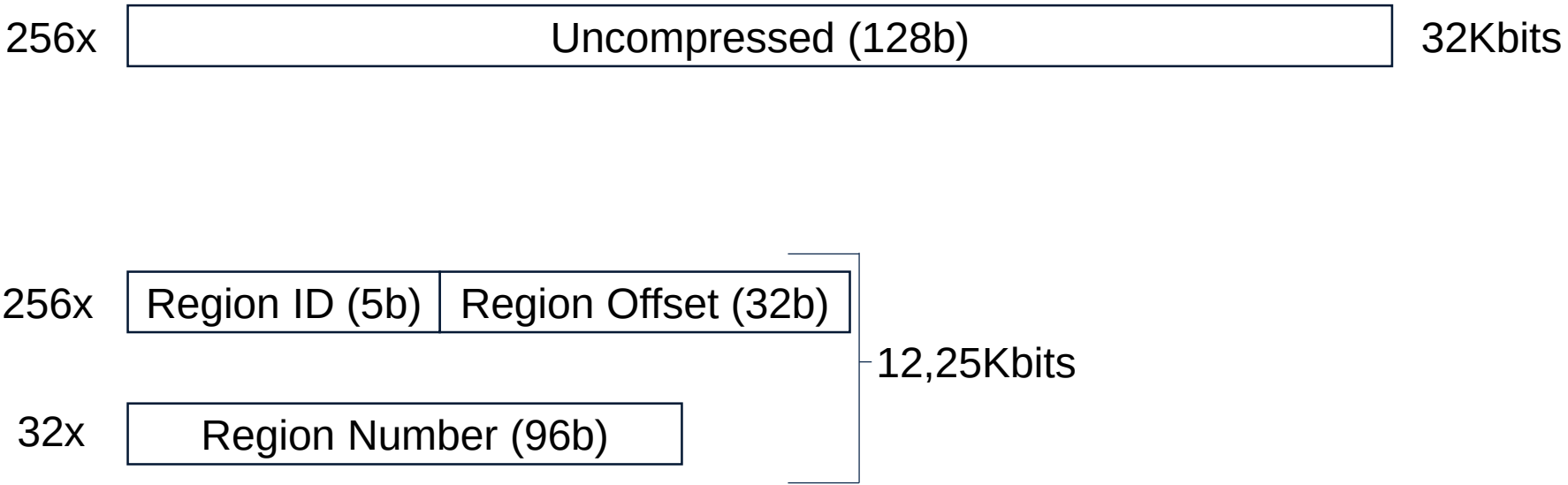
Region-based Compression

- **Decompression (e.g., PRF read into FU) -> Critical path**



Region-based Compression

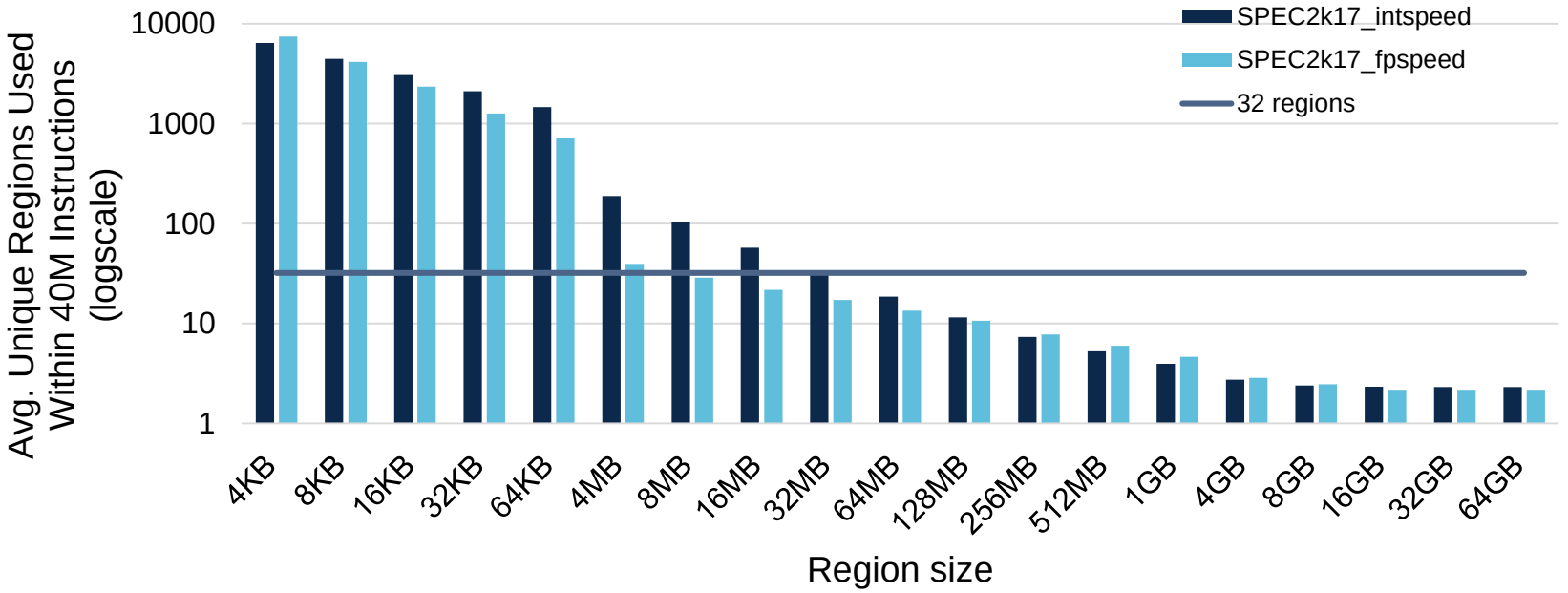
- **For example, assume 256 registers**
  - Uncompressed vs. 4GB regions (32 bits), 32 concurrent regions at any given time



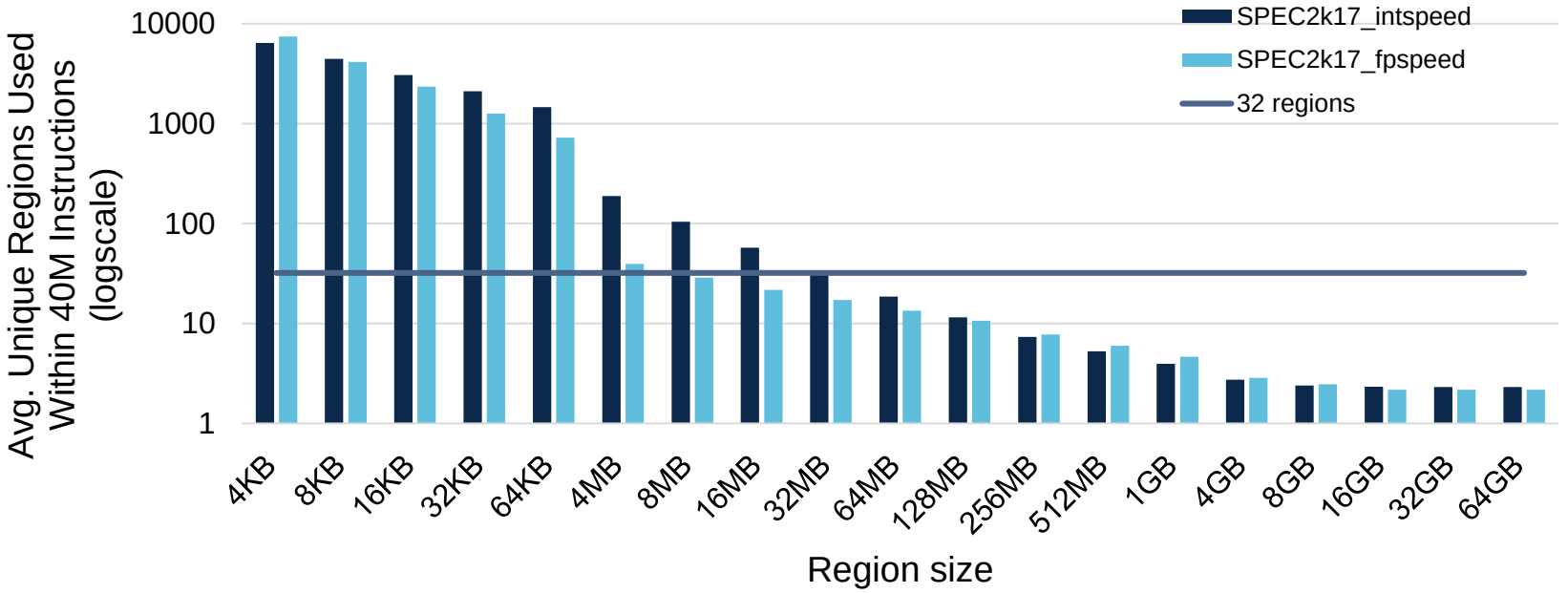
Compressed Physical Register (PRF)

- **Trade off PRF area for PRF latency**
  - Additional indexed read to get upper bits on PRF read
  - Additional associative search to match upper bits on PRF write
- **Number of regions and region size are designer choice**
  - Need at least 32 regions to be able to represent the architectural state
  - Smaller regions provide better savings
    - But may need to represent more concurrent regions at the same time
- **Use smallest region size where 32 concurrent regions are generally enough**
  - ➢ Make sure it is not going to cause too many reclamations

Compressed Physical Register (PRF)



Compressed Physical Register (PRF)



Baseline 128b => 32Kbits
Optimistic (32 MB - 25b regions) =>10,7Kbits
Pessimistic (64 GB - 36b regions) =>13,1Kbits

Compressed Physical Register (PRF)

- **Even very large region provide significant area savings**
  - 0.41x for 64GB regions
- **Can play the same game everywhere addresses are used**
  - Cache tags*
  - TLB tags and data
  - BTB tags* and targets
- **Can play some other games**
  - e.g., addition on compressed values is correct if carry is 0

*ISCA'96

128-bit CPU Microarchitecture Proposal – Summary

- **Reduce overhead by**
  - Clustering the backend, challenges:
    - Efficiently identifying instructions that need to go to the 128-bit cluster
    - Correctly identify what operators need to be implemented vs. emulated
  - Compressing values and tags where possible, challenge:
    - Trading of saved area for – potentially – increased latency on reads/writes

128-bit CPU Microarchitecture – Some « Crazy » Ideas

- **Less out-of-order?**
  - Baseline clustering steers along dependency chains or just round robin(-ish)
  - Here we steer based on what kind of data is manipulated
    - Do general purpose computations need OoO?
      - 64-bit InO + 128-bit OoO
      - This would be applicable to 32- and 64-bit machine, if it works

128-bit CPU Microarchitecture – Some « Crazy » Ideas

- **Less out-of-order?**
  - Baseline clustering steers along dependency chains or just round robin(-ish)
  - Here we steer based on what kind of data is manipulated
    - Do general purpose computations need OoO?
      - ➢ 64-bit InO + 128-bit OoO
      - ➢ This would be applicable to 32- and 64-bit machine, if it works

- **Leverage RISC-V to add helpful instructions**
  - *load/store pointer* instructions
  - Build a cache dedicated to pointers: 16B cachelines
    - Both tags and data are addresses, compress !
    - Limits impact of larger pointer on L1D cache footprint
    - But now I have holes in my L1D, depending on how software lays out structures
      - ✓ And many other issues ☺

128-bit Machines – That’s all Folks

Questions ?

SLS team https://tima.univ-grenoble-alpes.fr/research/sls
TIMA Lab https://tima.univ-grenoble-alpes.fr/