

# Summary of continued work on energy profiling

Mikael Hirki

June 8, 2015

# About me

- I'm a Master's degree student at Aalto (CS program)
- I have prior experience in performance profiling and optimization
- I'm doing my thesis on energy profiling

# Outline

- Step 1: RAPL validation
- Step 2: ParFullCMS energy trace analysis
- Step 3: Power modeling

# About the hardware

- I'm using a Haswell desktop computer
  - Intel Core i7-4770 @ 3.4 GHz
  - Intel DH87RL desktop board
  - 2 x 8 GB DDR3 1600 MHz
- Hyperthreading enabled
- Turbo Boost disabled

# RAPL findings

- The time between RAPL updates is 1 ms (+/- 20  $\mu$ s)
- Haswell measures energy in units of 61  $\mu$ J
- Latency of reading one RAPL counter using PAPI is about 550 nanoseconds
- Idle power consumption of my system is fairly low
  - Package (PKG): 0.95 W
  - Power plane 0 (PP0) / Processor cores: 18.4 mW
  - Power plane 1 (PP1) / Integrated graphics: 10 mW
  - DRAM: 1.6 W

# STREAM benchmark

- My results were very similar to what Filip discovered
- STREAM stresses DRAM
  - Highest bandwidth 14.7 GB/s
  - DRAM power consumption is only about 5.5W

# ParFullCMS

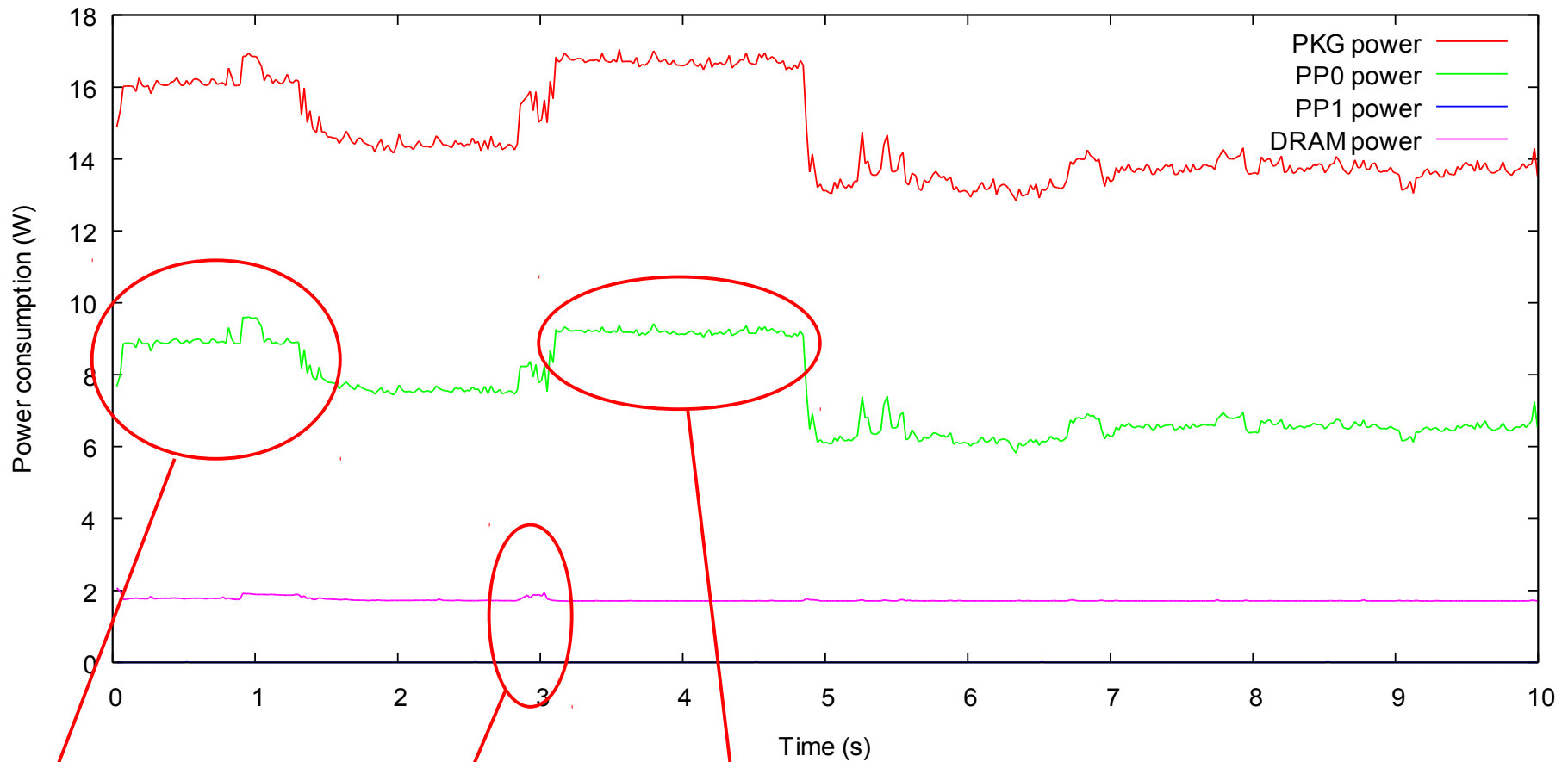
- I installed ParFullCMS from <http://davidlt.web.cern.ch/davidlt/g4parfullcms/G4test-src-V11.tar.gz>
- Running in single-threaded mode for energy profiling

# Graphical look at ParFullCMS

- I wrote my own tool to measure power consumption every 5 ms
  - This corresponds to how the energy profiling module works
- No stack traces
- Plotted the energy trace using Gnuplot



# ParFullCMS initialization (single-threaded)

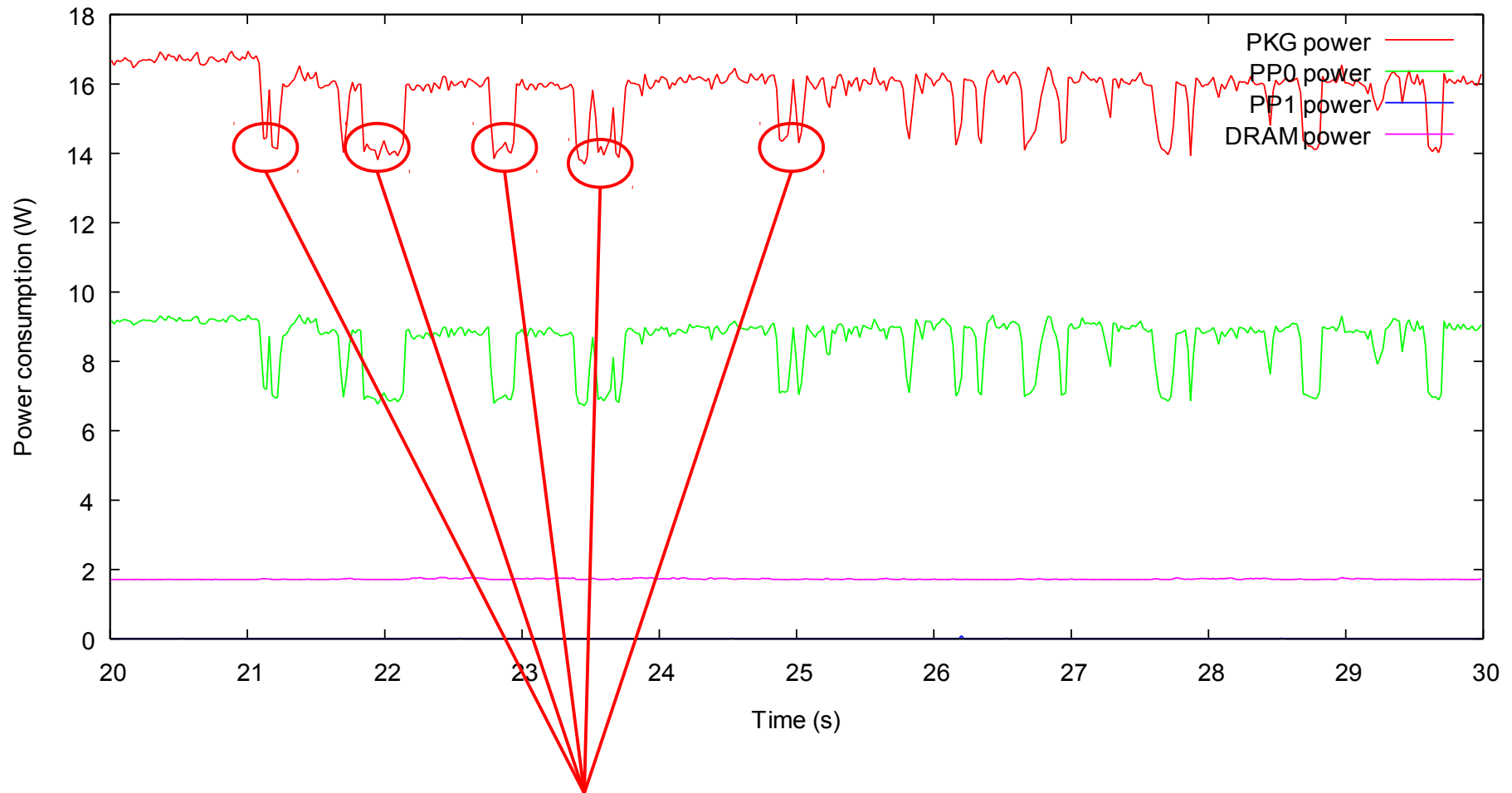


XML data parsing

DRAM access burst

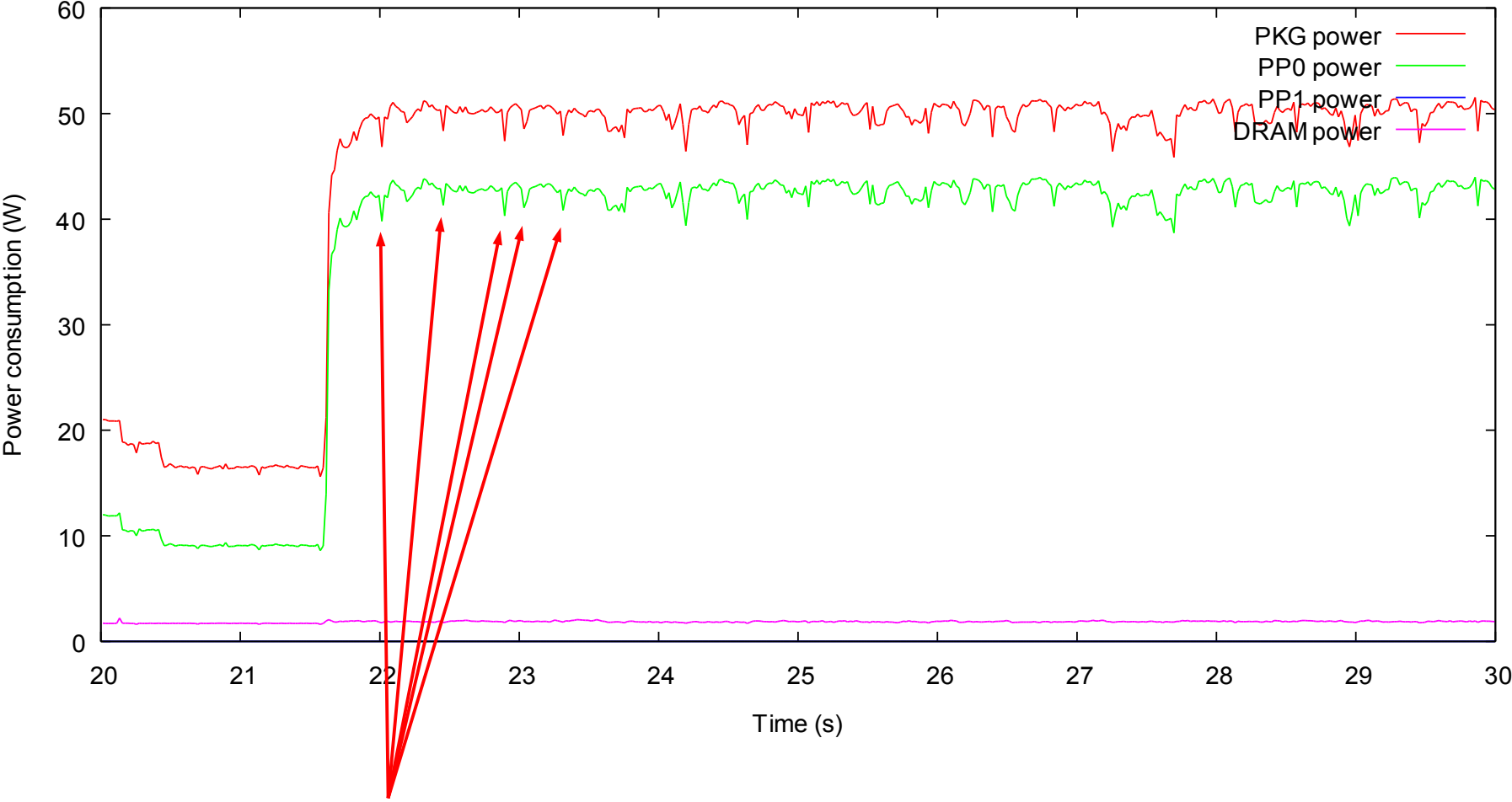
High power consumption caused by a recursive function

# ParFullCMS worker thread (single-threaded)



Strange drops in  
power consumption

# ParFullCMS worker threads (8 threads)



Similar drops in power consumption

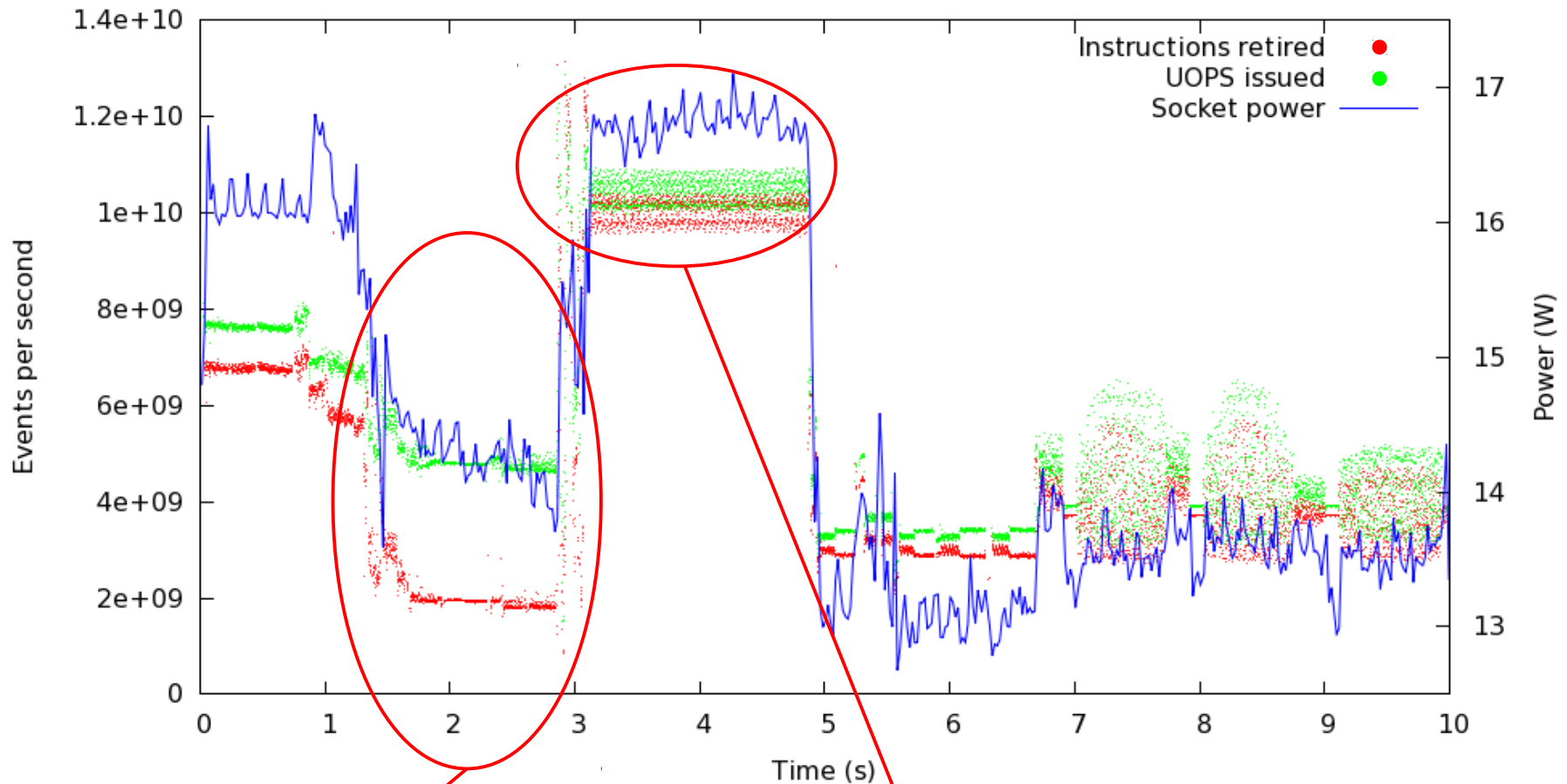
# Performance counters on Intel hardware

- 8 programmable counters per core
  - 4 counters per thread when using hyperthreading
- 3 fixed-function counters
  - Unhalted core cycles
    - Unhalted = core is not sleeping
    - Core cycles is affected by frequency scaling
  - Unhalted reference cycles (not available in perf)
  - Instructions retired

# μops issued

- Known as UOPS\_ISSUED.ANY in Intel's manual
- The number of micro-operations issued by the front-end to the back-end inside the processor
  - UOPS\_ISSUED includes speculative execution
  - Failed branch prediction increases UOPS\_ISSUED
  - Conceptually, this is much closer to the actual amount of work done than the number of instructions retired

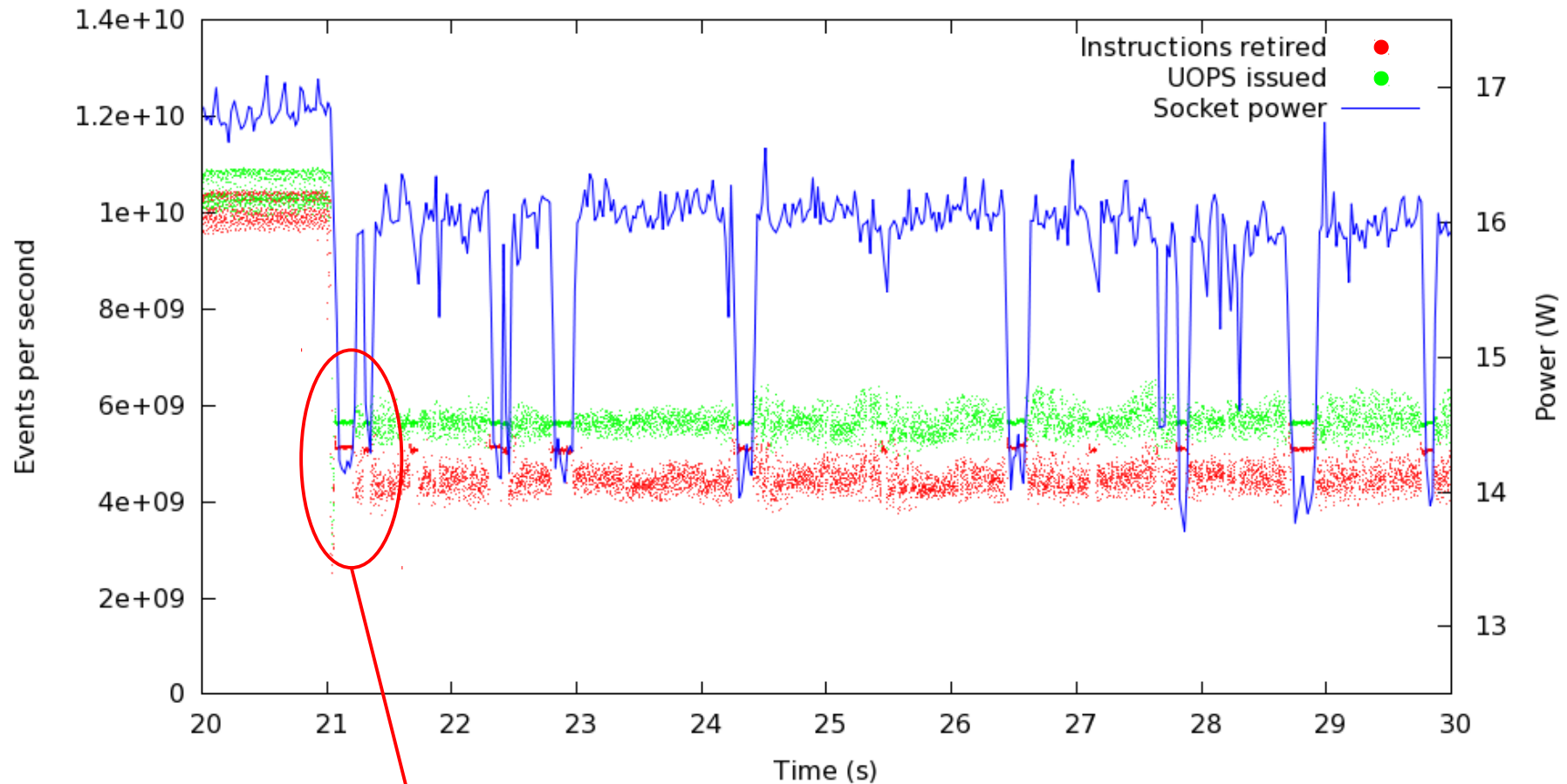
# Socket power compared to instructions retired and $\mu$ ops issued



$\mu$ ops issued shows much better correlation than instructions retired

High power consumption caused by a recursive function

# Socket power compared to instructions retired and $\mu$ ops issued



The mean number of  $\mu$ ops remains the same while power consumption drops

# Recursive function code

```
void G4ProductionCutsTable::ScanAndSetCouple(G4LogicalVolume* aLV,
                                             G4MaterialCutsCouple* aCouple,
                                             G4Region* aRegion)
{
    //Check whether or not this logical volume belongs to the same region
    if((aRegion!=0) && aLV->GetRegion()!=aRegion) return;

    //Check if this particular volume has a material matched to the couple
    if(aLV->GetMaterial()==aCouple->GetMaterial()) {
        aLV->SetMaterialCutsCouple(aCouple);
    }

    size_t noDaughters = aLV->GetNoDaughters();
    if(noDaughters==0) return;

    //Loop over daughters with same region
    for(size_t i=0;i<noDaughters;i++){
        G4LogicalVolume* daughterLVol = aLV->GetDaughter(i)->GetLogicalVolume();
        ScanAndSetCouple(daughterLVol,aCouple,aRegion);
    }
}
```

Recursive call



# Recursive function optimizations

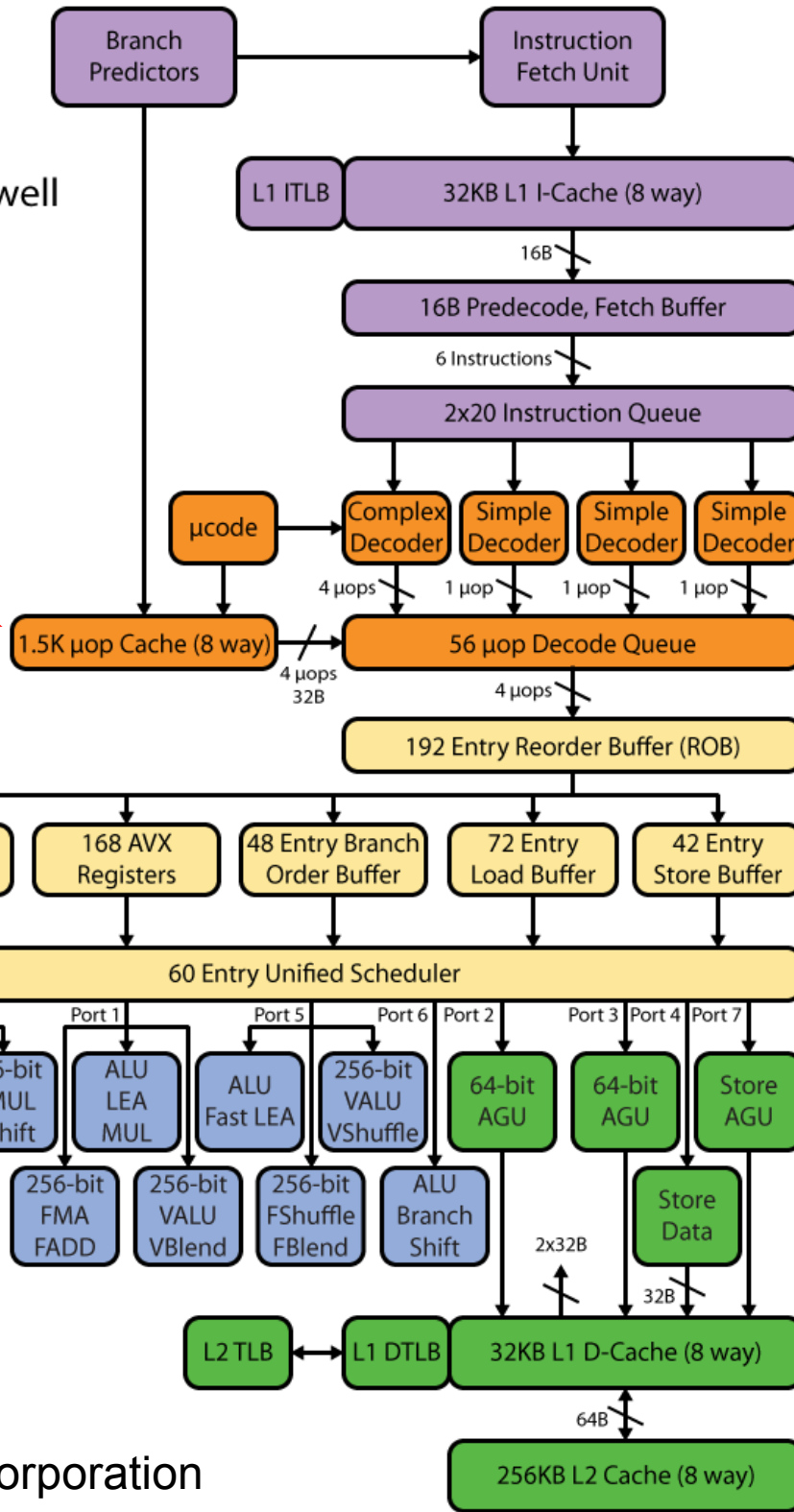
- Eliminating register spilling boosted performance by 13%
  - Register pressure caused by function arguments, and implicit 64bit  $\leftrightarrow$  32bit integer casts
- Power consumption remained the same
  - Thus energy consumption was also reduced by 13%

Haswell

Micro-operation cache

Instruction decoding pipeline

Backend execution ports



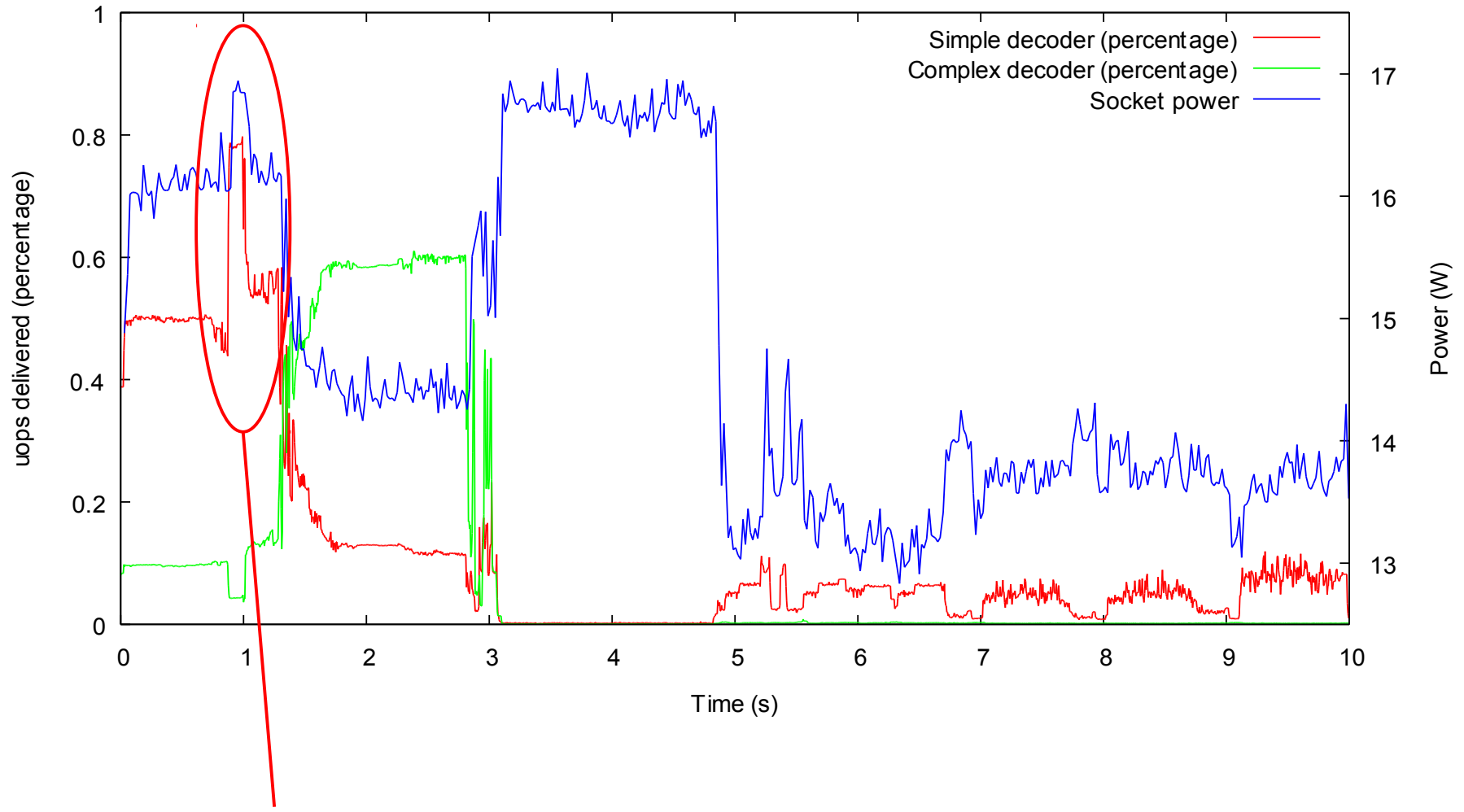
# Sources of micro-operations

- Three different sources of micro-operations in the frontend
  - Simple instruction decoders
  - Complex instruction decoders
    - Micro-operations are stored in the microcode ROM
    - Instructions that generate more than 4  $\mu$ ops
  - Micro-operation cache
    - Complex instructions fill an entire cache line (6  $\mu$ ops)

# Instruction decoders events

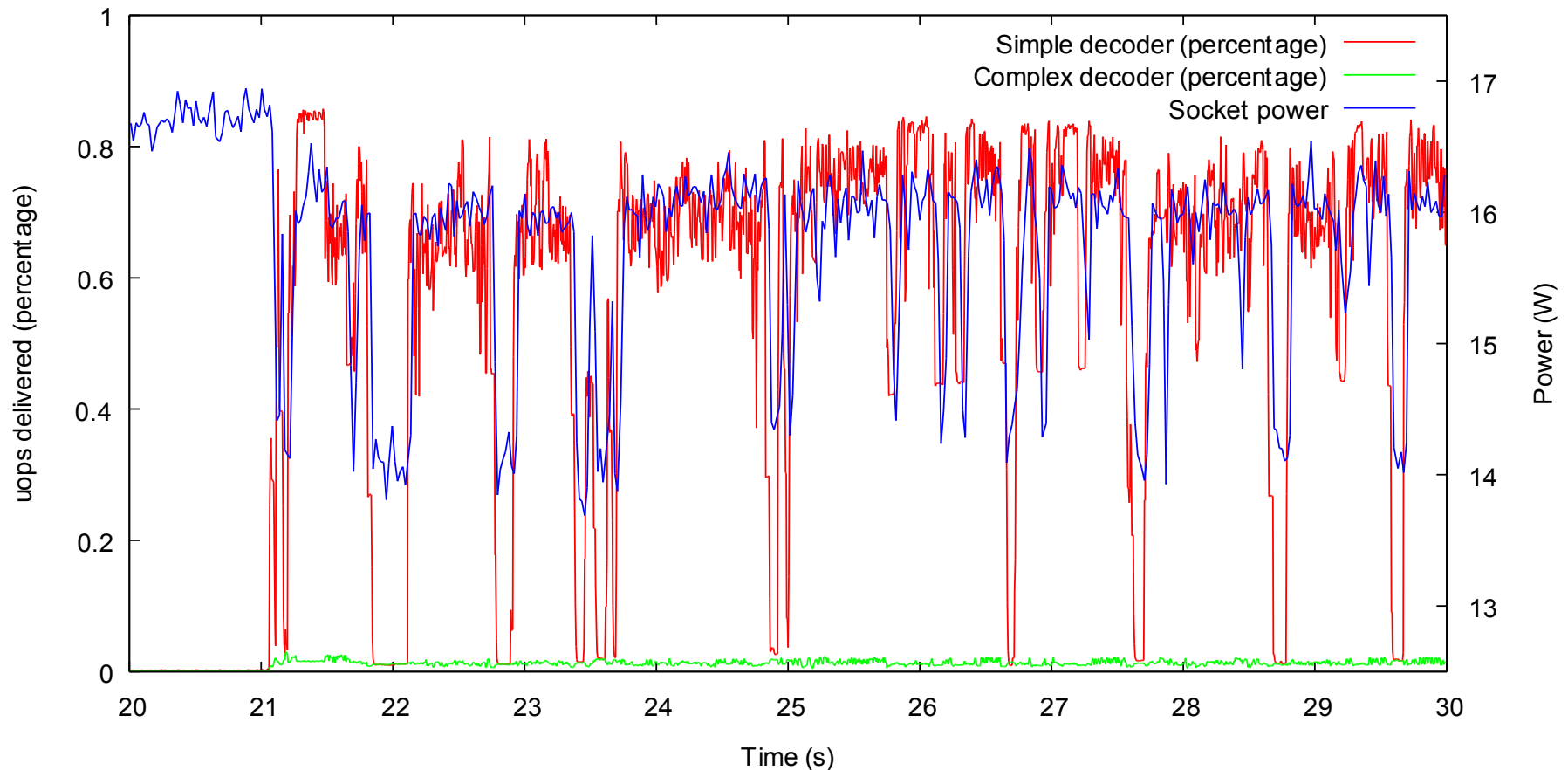
- IDQ.DSB\_UOPS
  - Micro-ops delivered from the micro-operation cache
- IDQ.MITE\_UOPS
  - Micro-ops delivered from the simple decoders
- IDQ.MS\_UOPS
  - Micro-ops delivered from the complex decoder (microsequencer)
- Typical distribution is 80% cached micro-operations, 15% from simple decoders and 5% from complex decoder

# Instruction decoder events graphically in ParFullCMS



Spike in power consumption correlates with instruction decoder activity

# Instruction decoder events graphically in ParFullCMS



The extra 2 watts of power correlates with instruction decoder activity

# Conclusion

- Instruction decoder consumes about 2 watts of power
- This is a power saving opportunity
  - Need to increase code locality
    - Do as much useful work as possible once code has been decoded into the  $\mu$ op cache
  - Need to optimize for smaller code size
    - Avoid using `-O3` and `-funroll-loops` too much

# Ideas for further work

- Power modeling
  - Currently in progress
- Compiler optimization comparison
- Intel vs ARM hardware comparison



# Power modeling

- ParFullCMS power consumption could be modeled using only two performance counters:
  - UOPS\_ISSUED.ANY
    - This accounts for the work done by the CPU backend
  - IDQ.MITE\_UOPS
    - This accounts for the (simple) instruction decoders
- This simple model fails to predict the power consumption in the PARSEC benchmark
  - Need to include L1 & L2 cache accesses and number of active cores in the model

# Extra slides

# Machine code view (first part)

| #   | Ir          | Hex                  | Assembly Instructions                                 |
|---|-------------|----------------------|---|
| 33 226C   |             | 0f 1f 40 00          | nopl 0x0(%rax)  |
| 33 2270   | 699 586 200 | 55                   | push %rbp   |
| 33 2271   | 699 586 200 | 48 89 e5             | mov %rsp,%rbp   |
| 33 2274   | 699 586 200 | 4c 89 65 e0          | mov %r12,-0x20(%rbp)                                  |
| 33 2278   | 699 586 200 | 4c 89 6d e8          | mov %r13,-0x18(%rbp)                                  |
| 33 227C   | 699 586 200 | 49 89 f4             | mov %rsi,%r12   |
| 33 227F   | 699 586 200 | 4c 89 75 f0          | mov %r14,-0x10(%rbp)                                  |
| 33 2283   | 699 586 200 | 48 89 5d d8          | mov %rbx,-0x28(%rbp)                                  |
| 33 2287   | 699 586 200 | 49 89 d6             | mov %rdx,%r14   |
| 33 228A   | 699 586 200 | 4c 89 7d f8          | mov %r15,-0x8(%rbp)                                   |
| 33 228E   | 699 586 200 | 48 83 ec 40          | sub \$0x40,%rsp                                       |
| 33 2292   | 699 586 200 | 48 85 c9             | test %rcx,%rcx  |
| 33 2295   | 699 586 200 | 48 89 7d c8          | mov %rdi,-0x38(%rbp)                                  |
| 33 2299   | 699 586 200 | 49 89 cd             | mov %rcx,%r13   |
| 33 229C   | 699 586 200 | 74 22                | je 3322c0 <G4ProductionCutsTable::ScanAndSetCouple... |
| 33 229E   | 699 586 200 | 48 3b 4e 48          | cmp 0x48(%rsi),%rcx                                   |
| 33 22A2   | 699 586 200 | 74 1c                | je 3322c0 <G4ProductionCutsTable::ScanAndSetCouple... |
| lump 699 586 200 of 699 586 200 times to 0x3322C0 |             |                      |   |
| 33 22A4   | 699 586 200 | 48 8b 5d d8          | mov -0x28(%rbp),%rbx                                  |
| 33 22A8   | 699 586 200 | 4c 8b 65 e0          | mov -0x20(%rbp),%r12                                  |
| 33 22AC   | 699 586 200 | 4c 8b 6d e8          | mov -0x18(%rbp),%r13                                  |
| 33 22B0   | 699 586 200 | 4c 8b 75 f0          | mov -0x10(%rbp),%r14                                  |
| 33 22B4   | 699 586 200 | 4c 8b 7d f8          | mov -0x8(%rbp),%r15                                   |
| 33 22B8   | 699 586 200 | c9                   | leaveq  |
| 33 22B9   | 699 586 200 | c3                   | retq  |
| 33 22BA   |             | 66 0f 1f 44 00 00    | nopw 0x0(%rax,%rax,1)                                 |
| 33 22C0   | 699 586 200 | 49 63 44 24 5c       | movslq 0x5c(%r12),%rax                                |
| 33 22C5   | 699 586 200 | 48 8b 15 3c ed fd 00 | mov 0xfded3c(%rip),%rdx # 1311008 <_DYNAMIC+...       |
| 33 22CC   | 699 586 200 | 48 8d 04 40          | lea (%rax,%rax,2),%rax                                |
| 33 22D0   | 699 586 200 | 48 c1 e0 04          | shl \$0x4,%rax  |
| 33 22D4   | 699 586 200 | 64 48 03 02          | add %fs:(%rdx),%rax                                   |
| 33 22D8   | 699 586 200 | 49 8b 56 10          | mov 0x10(%r14),%rdx                                   |
| 33 22DC   | 699 586 200 | 48 39 50 18          | cmp %rdx,0x18(%rax)                                   |
| 33 22E0   | 699 586 200 | 74 4e                | je 332330 <G4ProductionCutsTable::ScanAndSetCouple... |
| jump 2 331 954 of 699 586 200 times to 0x332330   |             |                      |   |

Branch never taken

Branch always taken

Lots of values being pushed onto the stack

Static branch prediction hint is wrong

Five registers are restored before return from this function

# Machine code view (first part)

| #   | Ir          | Hex                  | Assembly Instructions                                 |
|---|-------------|----------------------|---|
| 33 226C   |             | 0f 1f 40 00          | nopl 0x0(%rax)  |
| 33 2270   | 699 586 200 | 55                   | push %rbp   |
| 33 2271   | 699 586 200 | 48 89 e5             | mov %rsp,%rbp   |
| 33 2274   | 699 586 200 | 4c 89 65 e0          | mov %r12,-0x20(%rbp)                                  |
| 33 2278   | 699 586 200 | 4c 89 6d e8          | mov %r13,-0x18(%rbp)                                  |
| 33 227C   | 699 586 200 | 49 89 f4             | mov %rsi,%r12   |
| 33 227F   | 699 586 200 | 4c 89 75 f0          | mov %r14,-0x10(%rbp)                                  |
| 33 2283   | 699 586 200 | 48 89 5d d8          | mov %rbx,-0x28(%rbp)                                  |
| 33 2287   | 699 586 200 | 49 89 d6             | mov %rdx,%r14   |
| 33 228A   | 699 586 200 | 4c 89 7d f8          | mov %r15,-0x8(%rbp)                                   |
| 33 228E   | 699 586 200 | 48 83 ec 40          | sub \$0x40,%rsp                                       |
| 33 2292   | 699 586 200 | 48 85 c9             | test %rcx,%rcx  |
| 33 2295   | 699 586 200 | 48 89 7d c8          | mov %rdi,-0x38(%rbp)                                  |
| 33 2299   | 699 586 200 | 49 89 cd             | mov %rcx,%r13   |
| 33 229C   | 699 586 200 | 74 22                | je 3322c0 <G4ProductionCutsTable::ScanAndSetCouple... |
| 33 229E   | 699 586 200 | 48 3b 4e 48          | cmp 0x48(%rsi),%rcx                                   |
| 33 22A2   | 699 586 200 | 74 1c                | je 3322c0 <G4ProductionCutsTable::ScanAndSetCouple... |
| Jump 699 586 200 of 699 586 200 times to 0x3322C0 |             |                      |   |
| 33 22A4   | 699 586 200 | 48 8b 5d d8          | mov -0x28(%rbp),%rbx                                  |
| 33 22A8   | 699 586 200 | 4c 8b 65 e0          | mov -0x20(%rbp),%r12                                  |
| 33 22AC   | 699 586 200 | 4c 8b 6d e8          | mov -0x18(%rbp),%r13                                  |
| 33 22B0   | 699 586 200 | 4c 8b 75 f0          | mov -0x10(%rbp),%r14                                  |
| 33 22B4   | 699 586 200 | 4c 8b 7d f8          | mov -0x8(%rbp),%r15                                   |
| 33 22B8   | 699 586 200 | c9                   | leaveq  |
| 33 22B9   | 699 586 200 | c3                   | retq  |
| 33 22BA   | 699 586 200 | 66 0f 3f 44 00 00    | nopw 0x0(%rax,%rax,1)                                 |
| 33 22C0   | 699 586 200 | 49 63 44 24 5c       | movsld 0x5c(%r12),%rax                                |
| 33 22C5   | 699 586 200 | 48 8b 15 3c ed fd 00 | mov 0xfded3c(%rip),%rdx # 1311008 <_DYNAMIC+...       |
| 33 22CC   | 699 586 200 | 48 8d 04 04          | lea (%rax,%rax,2),%rax                                |
| 33 22D0   | 699 586 200 | 48 c1 e0 04          | shl \$0x4,%rax  |
| 33 22D4   | 699 586 200 | 64 48 03 02          | add %fs,%rdx,%rax                                     |
| 33 22D8   | 699 586 200 | 49 8b 56 10          | mov 0x10(%r14),%rdx                                   |
| 33 22DC   | 699 586 200 | 48 39 56 18          | cmp %rdx,0x18(%rax)                                   |
| 33 22E0   | 699 586 200 | 74 4e                | je 332330 <G4ProductionCutsTable::ScanAndSetCouple... |
| Jump 2 331 954 of 699 586 200 times to 0x332330   |             |                      |   |

Dynamic constant fetched from memory

Code for aLV->GetMaterial()

%fs is a segmentation register! It's used for thread-local storage on x86-64

C++ tends to hide implementation details...

# Machine code view (second part)

| #       | lr          | Hex                  | Assembly Instructions   |
|---------|-------------|----------------------|---|
| 33 22E0 | 699 586 200 | 74 4e                | je 332330 <G4ProductionCutsTable::ScanAndSetCouple...<br>Jump 2 331 954 of 699 586 200 times to 0x332330                    |
| 33 22E2 | 699 586 200 | 49 8b 04 24          | mov (%r12),%rax   |
| 33 22E6 | 699 586 200 | 4d 8b 7c 24 08       | mov 0x8(%r12),%r15  |
| 33 22EB | 699 586 200 | 49 29 c7             | sub %rax,%r15   |
| 33 22EE | 699 586 200 | 49 c1 ff 03          | sar \$0x3,%r15  |
| 33 22F2 | 699 586 200 | 4d 63 ff             | movslq %r15d,%r15   |
| 33 22F5 | 699 586 200 | 4d 85 ff             | test %r15,%r15  |
| 33 22F8 | 699 586 200 | 74 aa                | je 3322a4 <G4ProductionCutsTable::ScanAndSetCouple...<br>Jump 540 495 000 of 699 586 200 times to 0x3322A4                  |
| 33 22FA | 159 091 200 | 31 db                | xor %ebx,%ebx   |
| 33 22FC | 159 091 200 | eb 06                | jmp 332304 <G4ProductionCutsTable::ScanAndSetCouple...<br>Jump 159 091 200 times to 0x332304                                |
| 33 22FE |             | 66 90                | xchg %ax,%ax  |
| 33 2300 | 540 494 400 | 49 8b 04 24          | mov (%r12),%rax   |
| 33 2304 | 699 585 600 | 48 63 d3             | movslq %ebx,%rdx  |
| 33 2307 | 699 585 600 | 48 8b 7d c8          | mov -0x38(%rbp),%rdi  |
| 33 230B | 699 585 600 | 4c 89 e9             | mov %r13,%rcx   |
| 33 230E | 699 585 600 | 48 8b 04 d0          | mov (%rax,%rdx,8),%rax  |
| 33 2312 | 699 585 600 | 48 83 c3 01          | add \$0x1,%rbx  |
| 33 2316 | 699 585 600 | 4c 89 f2             | mov %r14,%rdx   |
| 33 2319 | 699 585 600 | 48 8b 70 10          | mov 0x10(%rax),%rsi   |
| 33 231D | 699 585 600 | e8 16 61 fd ff       | callq 308438 <G4ProductionCutsTable::ScanAndSetCouple...<br>(cycle) 699585600 call(s) to 'G4ProductionCutsTable::ScanAndSel |
| 33 2322 | 699 585 600 | 49 39 df             | cmp %rbx,%r15   |
| 33 2325 | 699 585 600 | 77 d9                | ja 332300 <G4ProductionCutsTable::ScanAndSetCouple...<br>Jump 540 494 400 of 699 585 600 times to 0x332300                  |
| 33 2327 | 159 091 200 | e9 78 ff ff ff       | jmpq 3322a4 <G4ProductionCutsTable::ScanAndSetCouple...<br>Jump 159 091 200 times to 0x3322A4                               |
| 33 232C |             | 0f 1f 40 00          | nopl 0x0(%rax)  |
| 33 2330 | 2 331 954   | 4c 89 70 28          | mov %r14,0x28(%rax)   |
| 33 2334 | 2 331 954   | eb ac                | jmp 3322e2 <G4ProductionCutsTable::ScanAndSetCouple...<br>Jump 2 331 954 times to 0x3322E2                                  |
| 33 2336 |             | 66 2e 0f 1f 84 00 00 | nopw %cs:0x0(%rax,%rax,1)   |
| 33 233D |             | 00 00 00             |   |

Signed 32bit  
-> 64bit integer  
conversions

Load value  
from stack

Recursive call

# Optimization opportunities

```
void G4ProductionCutsTable::ScanAndSetCouple(G4LogicalVolume* aLV,
                                             G4MaterialCutsCouple* aCouple,
                                             G4Region* aRegion)
{
    //Check whether or not this logical volume belongs to the same region
    if((aRegion!=0) && aLV->GetRegion()!=aRegion) return;

    //Check if this particular volume has a material matched to the couple
    if(aLV->GetMaterial()==aCouple->GetMaterial()) {
        aLV->SetMaterialCutsCouple(aCouple);
    }

    size_t noDaughters = aLV->GetNoDaughters();
    if(noDaughters==0) return;

    //Loop over daughters with same region
    for(size_t i=0;i<noDaughters;i++){
        G4LogicalVolume* daughterLVol = aLV->GetDaughter(i)->GetLogicalVolume();
        ScanAndSetCouple(daughterLVol,aCouple,aRegion);
    }
}
```

Redundant check that should be done outside recursion

This is always true in the benchmark

This result should be cached since it's always the same

This early return is redundant but compiler already optimizes it nicely

Getting rid of the recursion

# Optimization opportunities

```
void G4ProductionCutsTable::ScanAndSetCouple(G4LogicalVolume* aLV,
                                             G4MaterialCutsCouple* aCouple,
                                             G4Region* aRegion)
{
    //Check whether or not this logical volume belongs to the same region
    if((aRegion!=0) && aLV->GetRegion()!=aRegion) return;

    //Check if this particular volume has a material matched to the couple
    if(aLV->GetMaterial()==aCouple->GetMaterial()) {
        aLV->SetMaterialCutsCouple(aCouple);
    }

    size_t noDaughters = aLV->GetNoDaughters();
    if(noDaughters==0) return;

    //Loop over daughters with same region
    for(size_t i=0;i<noDaughters;i++){
        G4LogicalVolume* daughterLVol = aLV->GetDaughter(i)->GetLogicalVolume();
        ScanAndSetCouple(daughterLVol,aCouple,aRegion);
    }
}
```

size\_t is unsigned 64-bit integer

32-bit integer returned, implicit cast to 64-bit

GetDaughter takes a 32-bit integer as the parameter, implicit cast to 32-bit