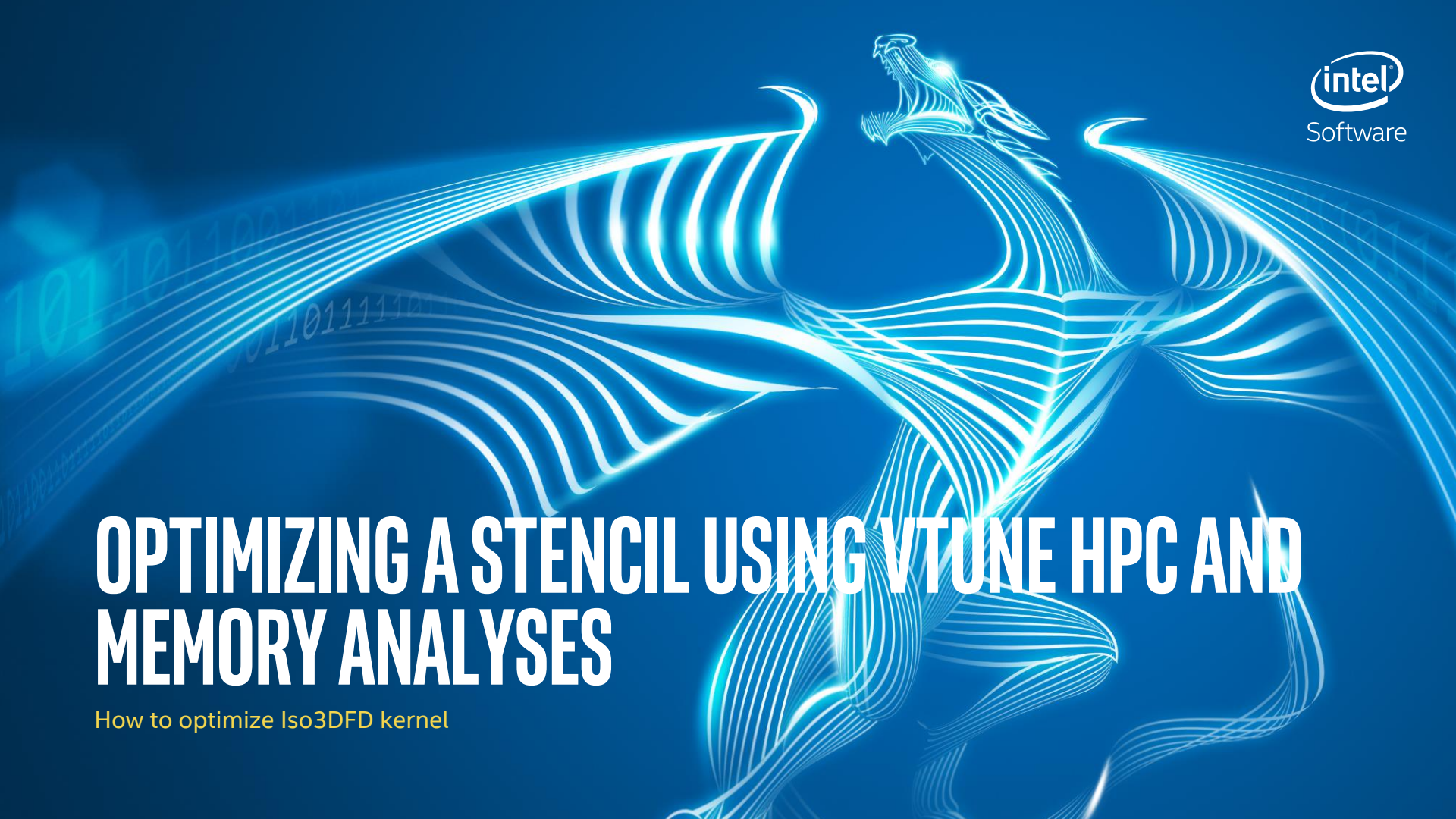


OPTIMIZING A STENCIL USING VTUNE HPC AND MEMORY ANALYSES

How to optimize Iso3DFD kernel

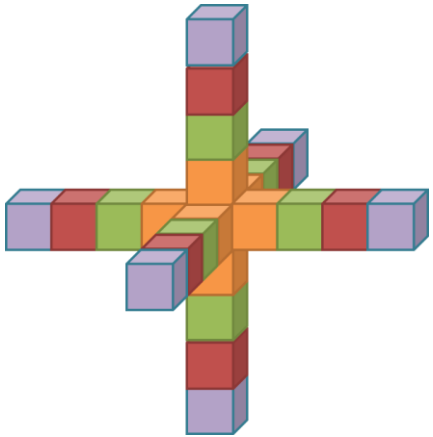
AGENDA

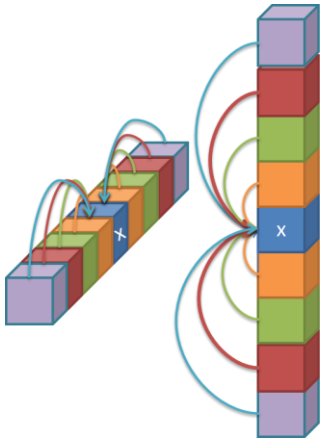
- What is iso3DFD
- Brief introduction to Vtune Amplifier and where to start ?
- Dev00
- Dev01 – Threading
- Dev02 – Memory optimization
- Dev03 – Cache blocking
- Dev04 - Vectorization
- Dev05 – First touch and NUMA effect

WHAT IS ISO3DFD ?

HOW DOES A STENCIL WORK ?

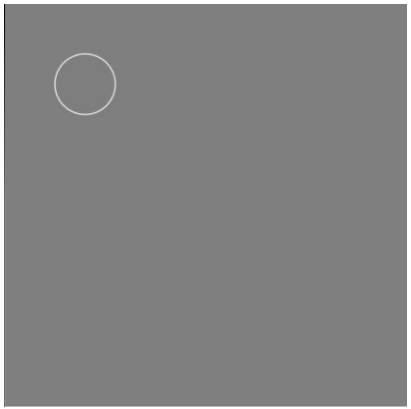
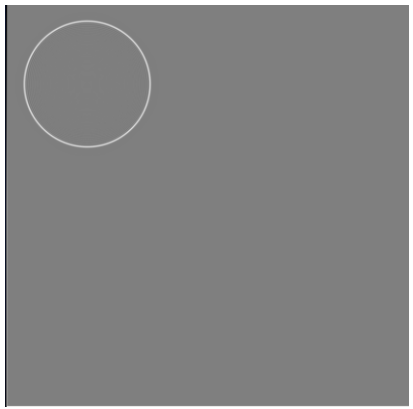
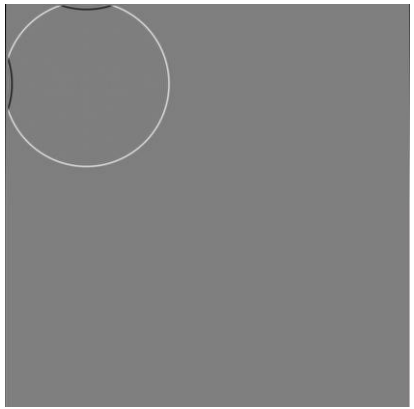
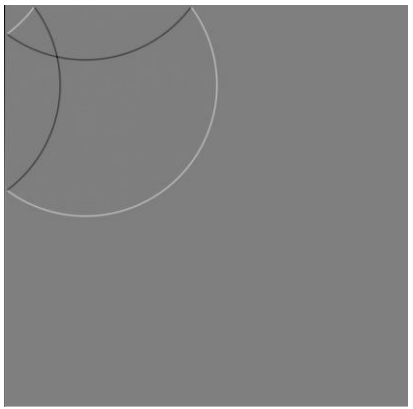
- For computing $P_{t+1}(x,y,z)$, we need to use all neighbors in the 3 dimensions of $P_t(x,y,z)$.
- The stencil looks like a 3D cross in Iso3DFD





ISO3DFD 2D CUT

- This is a 2D cut of our 3D volume
- We create a perturbation and look at the pressure for 4 different time steps
- We see that there is no boundary condition

ISO3DFD – A WAVE PROPAGATION KERNEL

- 3D Finite Difference
- Acoustic isotropic, pressure only scheme,
- 16th order in space 2nd order in time
- No boundary conditions
- OpenMP (no MPI)

```
For (int iz=0; iz<n3; iz++)
For (int iy=0; iy<n2; iy++)
For (int ix=0; ix<n1; ix++) {
    int offset = iz*dimn1n2 + iy*n1 + ix;
    float value = 0.0;
    value += ptr_prev[offset]*coeff[0];
    for(int ir=1; ir<= 8 ; ir++) {
        value += coeff[ir] * (ptr_prev[offset + ir] + ptr_prev[offset - ir]);
        value += coeff[ir] * (ptr_prev[offset + ir*n1] + ptr_prev[offset - ir*n1]);
        value += coeff[ir] * (ptr_prev[offset + ir*dimn1n2] + ptr_prev[offset -
ir*dimn1n2]);
    }
    ptr_next[offset] = 2.0f* ptr_prev[offset] - ptr_next[offset] + value*ptr_vel[offset];
}
```

WHICH HARDWARE

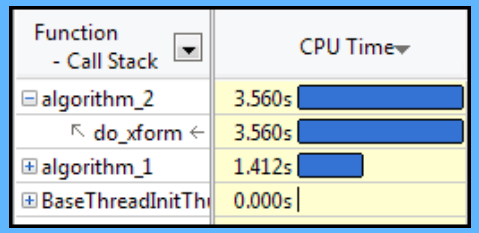
- For the rest of the presentation we are using a 2 sockets system
  - Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz
  - Each processor as 18 cores
  - Total of 36 cores
  - Haswell system with AVX2 enabled

BRIEF INTRODUCTION TO VTUNE AMPLIFIER AND WHERE TO START ?

VTUNE IS A VERY POWERFULL TOOL

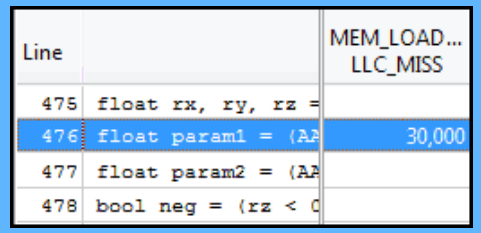
Where is my application…

Spending Time?



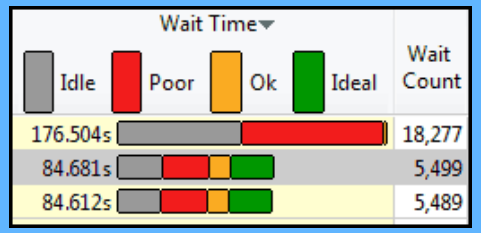
- Focus tuning on functions taking time
- See call stacks
- See time on source

Wasting Time?



- See cache misses on your source
- See functions sorted by # of cache misses

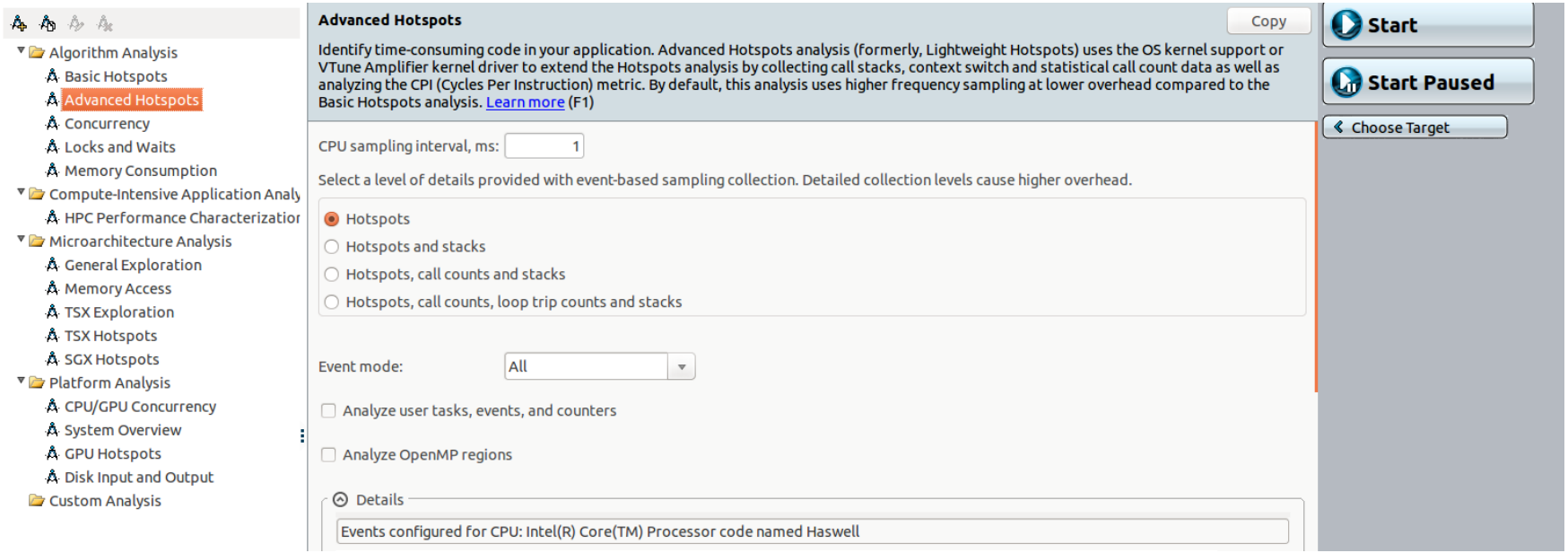
Waiting Too Long?



- See locks by wait time
- Red/Green for CPU utilization during wait

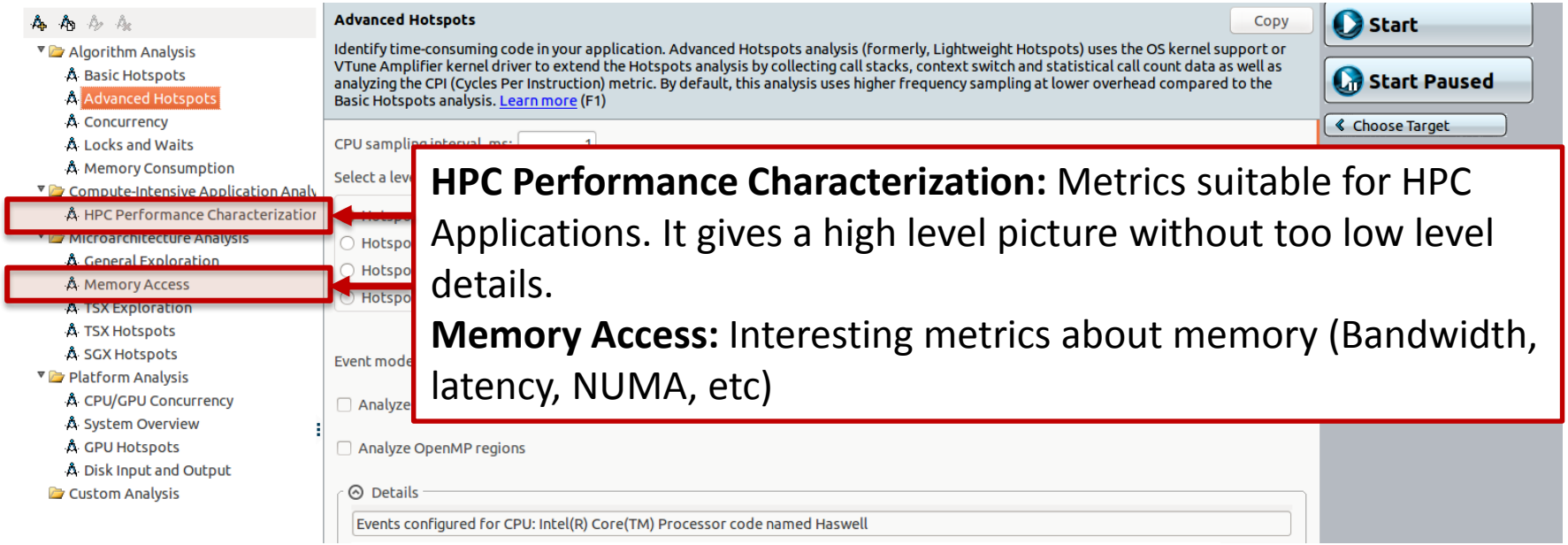
WHICH ANALYSIS SHOULD I RUN FIRST ?

- Historically we've encourage users to start with Hotspots or Advanced hotspots



WHICH ANALYSIS SHOULD I RUN FIRST ?

- But recently we added new analyses dedicated to HPC

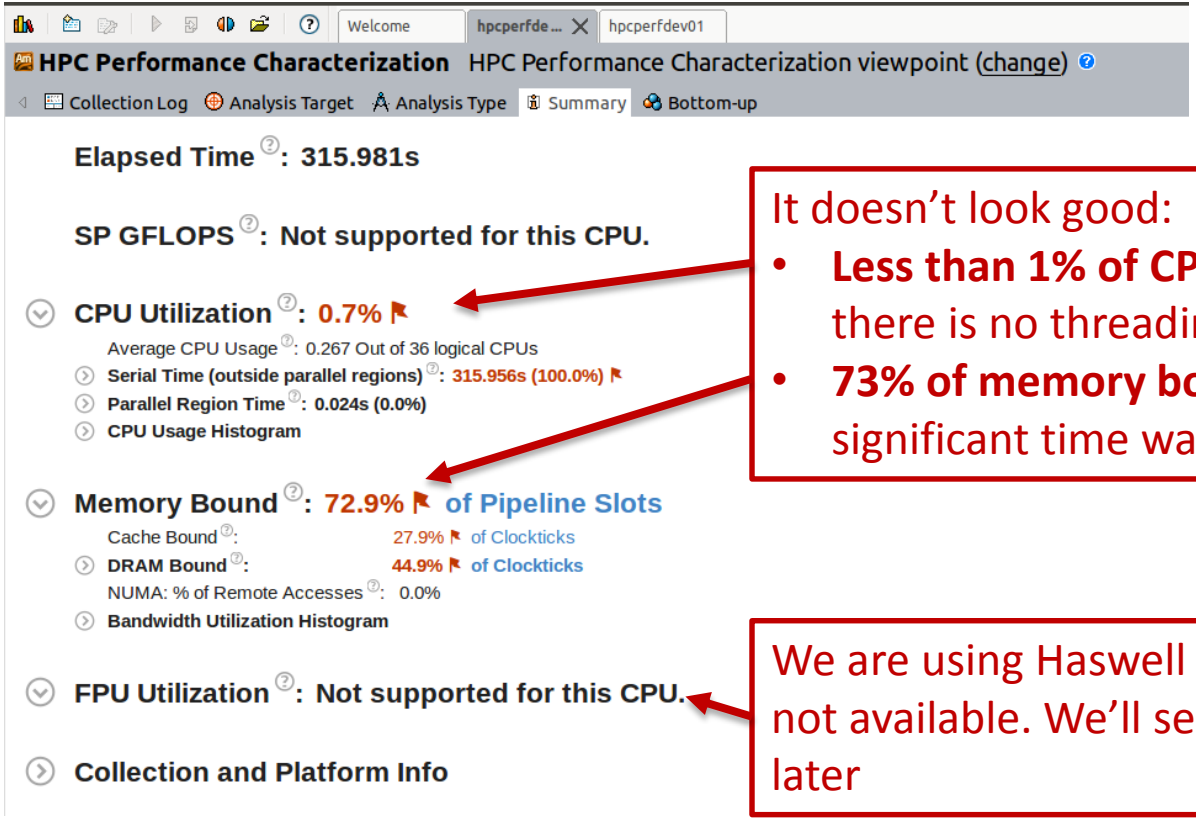


ISO3DFD DEV00 – A VERY UNOPTIMIZED VERSION

DEVOO_UNOPTIMIZED (WE SHOULD SAY SUB_OPTIMIZED ☺)

- We created a sub-optimized version of iso3DFD:
  - No parallelization
  - No vectorization
  - Sub-optimal memory accesses
- Let's see how Vtune can highlight the problems in this code
- 1st **step**, lets have an idea about the main bottlenecks
  - **Run the HPC Performance Characterization**

HPC PERFORMANCE CHARACTERIZATION ON DEVOO



It doesn't look good:

- **Less than 1% of CPU utilization** (it looks like there is no threading)
- **73% of memory bound**, we are spending some significant time waiting for data

We are using Haswell and unfortunately **Flops** are not available. We'll see how to solve this problem later

CPU WITH FLOPS COUTERS AVAILABLE

Memory Bound: 73.0% of Pipeline Slots

- Cache Bound: 2.2% of Clockticks
- **DRAM Bound**: 67.2% of Clockticks
- NUMA: % of Remote Accesses: 0.0%
- **Bandwidth Utilization Histogram**

FPU Utilization: 0.2%

- SP FLOPs per Cycle: 0.149 Out of 64
- Vector Capacity Usage: 6.2%
- FP Instruction Mix:
  - % of Packed FP Instr.: 0.0%
    - % of 128-bit: 0.0%
    - % of 256-bit: 0.0%
    - % of 512-bit: 0.0%
  - % of Scalar FP Instr.: 100.0%
- FP Arith/Mem Rd Instr. Ratio: 0.801
- FP Arith/Mem Wr Instr. Ratio: 53.713
- **Top Loops/Functions with FPU Usage by CPU Time**

This section provides information for the most time consuming loops/functions with floating point operations.

Function	CPU Time	FPU Utilization	Vector Instruction Set	Loop Type
[Loop at line 41 in iso_3dfd]	135.459s	0.2%		Body
[Loop at line 41 in iso_3dfd]	132.997s	0.2%		Body

This example was obtained on Skylake where flops counters are available

CPU UTILIZATION IN DETAILS

CPU Utilization [?]: **0.7%**

Average CPU Usage [?]: 0.267 Out of 36 logical CPUs

Serial Time (outside parallel regions) [?]: **315.956s (100.0%)**

Top Serial Hotspots (outside parallel regions)

This section lists the loops and functions executed serially in the master thread outside of any OpenMP region and consuming the most
metric includes the Wait time of the master thread, it may significantly exceed the aggregated CPU time in the table.

Function	Module	Serial CPU Time [?]
[Loop at line 41 in iso_3dfd]	iso3dfd_dev00_unoptimized_cpu_avx2.exe	42.194s
[Loop at line 41 in iso_3dfd]	iso3dfd_dev00_unoptimized_cpu_avx2.exe	41.279s
__do_softirq	vmlinux	0.320s
_spin_unlock_irqrestore	vmlinux	0.080s
clear_page_c_e	vmlinux	0.077s
[Others]		0.118s

Parallel Region Time [?]: **0.024s (0.0%)**

Estimated Ideal Time [?]: 0.002s (0.0%)

OpenMP Potential Gain [?]: 0.023s (0.0%)

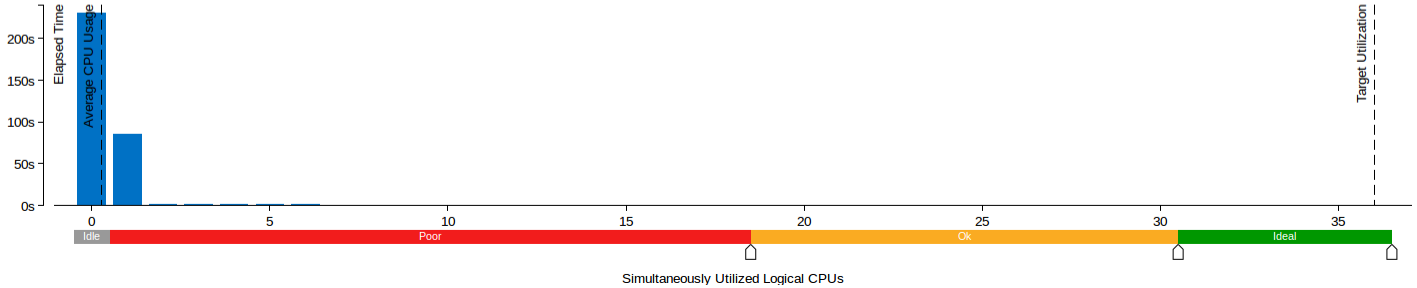
Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed

OpenMP Region	OpenMP Potential Gain [?]	(%) [?]	OpenMP Region Time [?]
mainompparallel:36@unknown:128:134	0.023s	0.0%	0.024s

CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.



- HPC Characterization indicates that CPU utilization is very low
  - Lets focus on it first
  - We see that only 1 thread is used instead of 36 !
  - It looks like we need to implement some threading

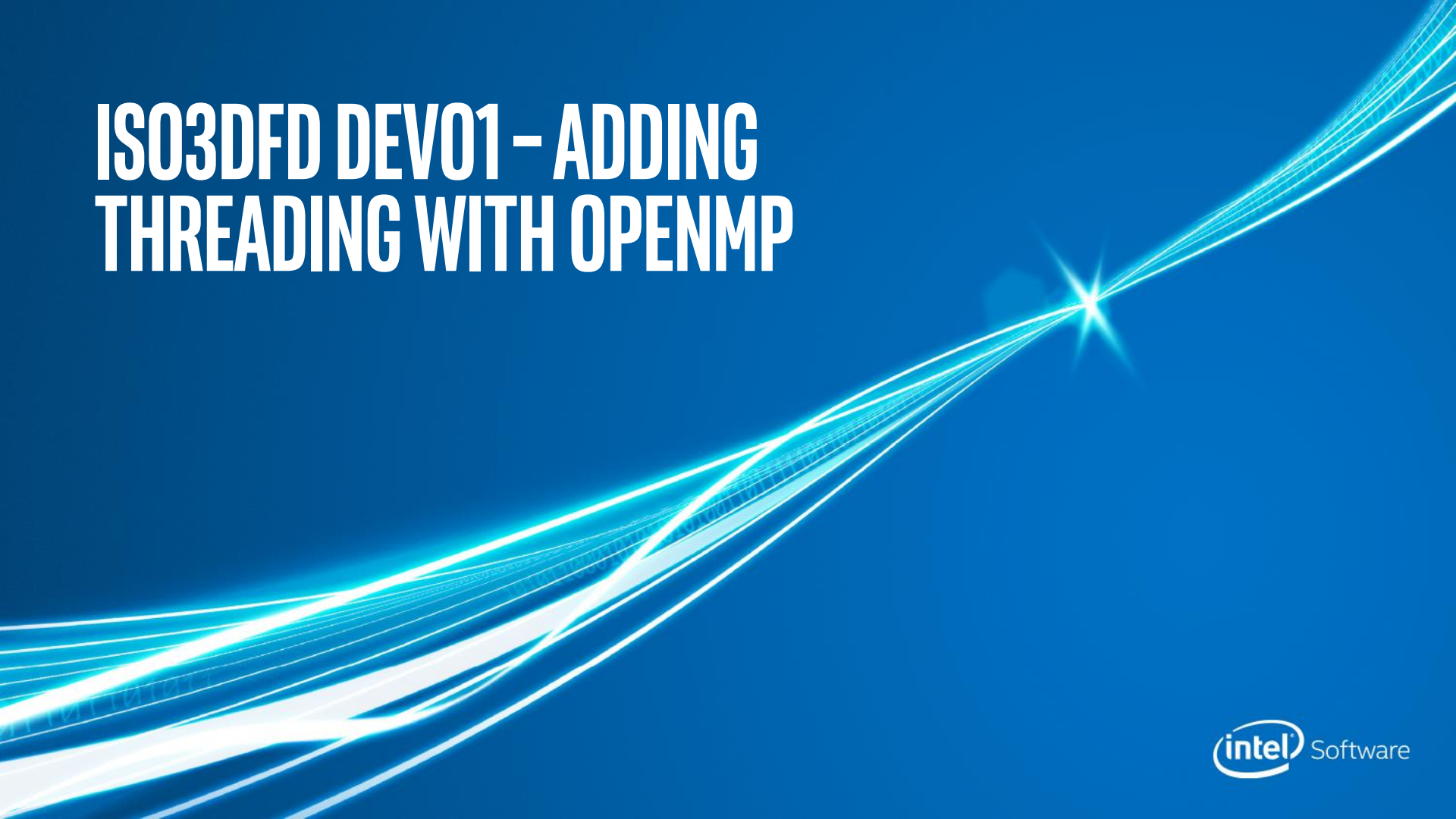
LETS LOOK AT THE CODE OF DEVOO

Adding parallelism can be done here

```
void iso_3dfd_it(float *ptr_next, float *ptr_prev, float *ptr_vel, float
*coeff, const int n1, const int n2, const int n3, const int
num_threads, const int n1_Tblock, const int n2_Tblock, const int
n3_Tblock){
    int dimn1n2 = n1*n2;//This value will be used later
    for(int ix=0; ix<n1; ix++) {
        for(int iy=0; iy<n2; iy++) {
            for(int iz=0; iz<n3; iz++) {
                … // compute the stencil
            }
        }
    }
}
```

Single time step iteration so no dependency inside those loops, threading is then possible

ISO3DFD DEV01 – ADDING THREADING WITH OPENMP



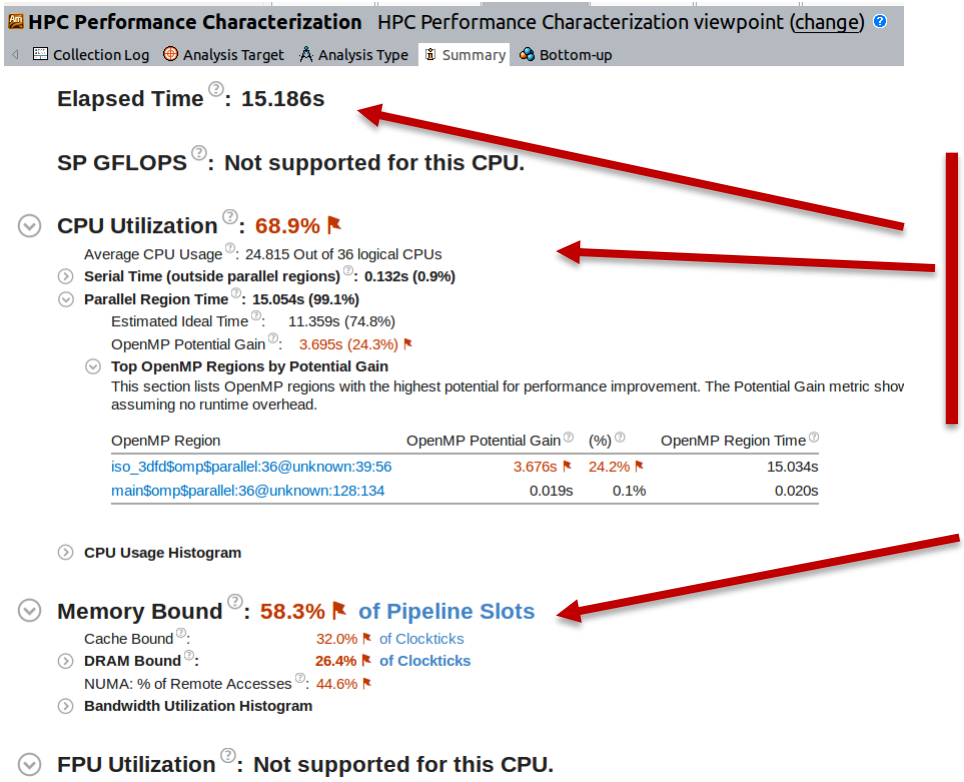
ADDING A SINGLE LINE OF OPENMP TO RUN ON ALL CORES

```
void iso_3dfd_it(float *ptr_next,  float *ptr_prev, float *ptr_vel,  float
*coeff, const int n1, const int n2, const int n3, const int
num_threads, const int n1_Tblock, const int n2_Tblock, const int
n3_Tblock){
    int dimn1n2 = n1*n2;//This value will be used later
    #pragma omp parallel for default(shared)
    for(int ix=0; ix<n1; ix++) {
        for(int iy=0; iy<n2; iy++) {
            for(int iz=0; iz<n3; iz++) {
                … // compute the stencil
            }
        }
    }
}
```

WHAT IS THE NEXT STEP ?

- We can iterate and run Vtune again to see what is the next bottleneck
  - Run another **HPC Performance Characterization**
  - Look for the next bottleneck
  - Try to optimize it
  - Iterate again
- Other tools can be used in the iteration process
  - **Intel® Advisor** is better suited for vectorization (FPU utilization)
  - **Performance Snapshot** can also be used instead of HPC characterization

HPC PERFORMANCE CHARACTERIZATION ON DEVCLOUD



- Threading is already much better
- We went from 316s to 15.2s (=~20X speedup)
- It looks like some improvement on threading can still be obtained

- The biggest bottleneck right now comes from memory

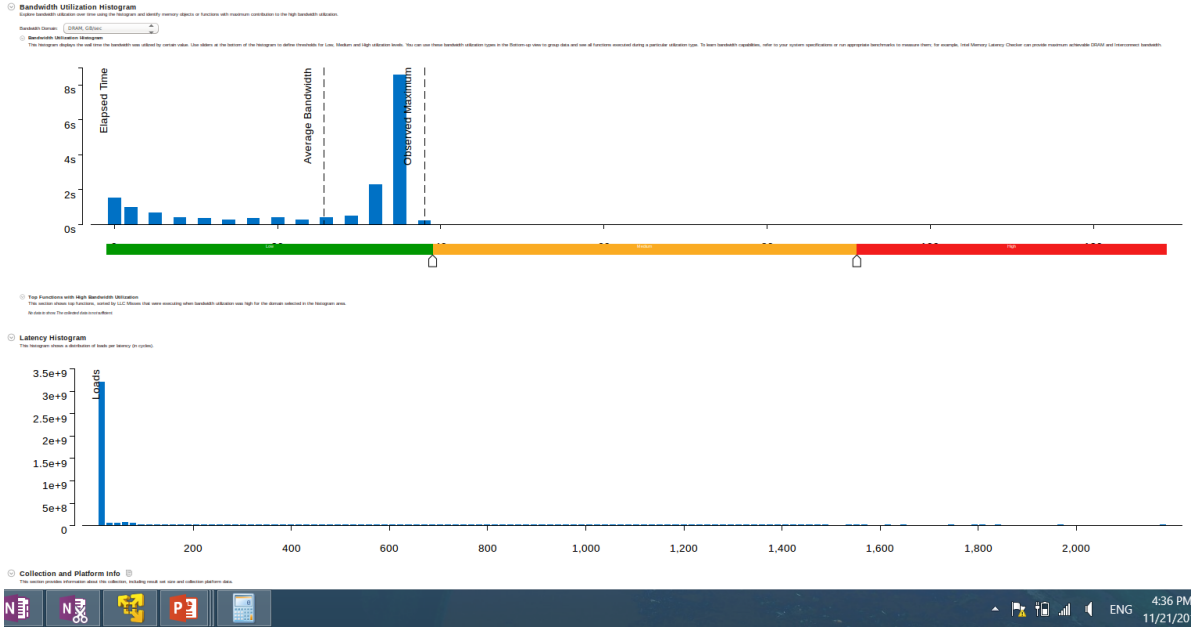
MEMORY ACCESS ANALYSIS IN VTUNE

Memory Access Memory Usage viewpoint (change)

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | P

Elapsed Time: 17.374s

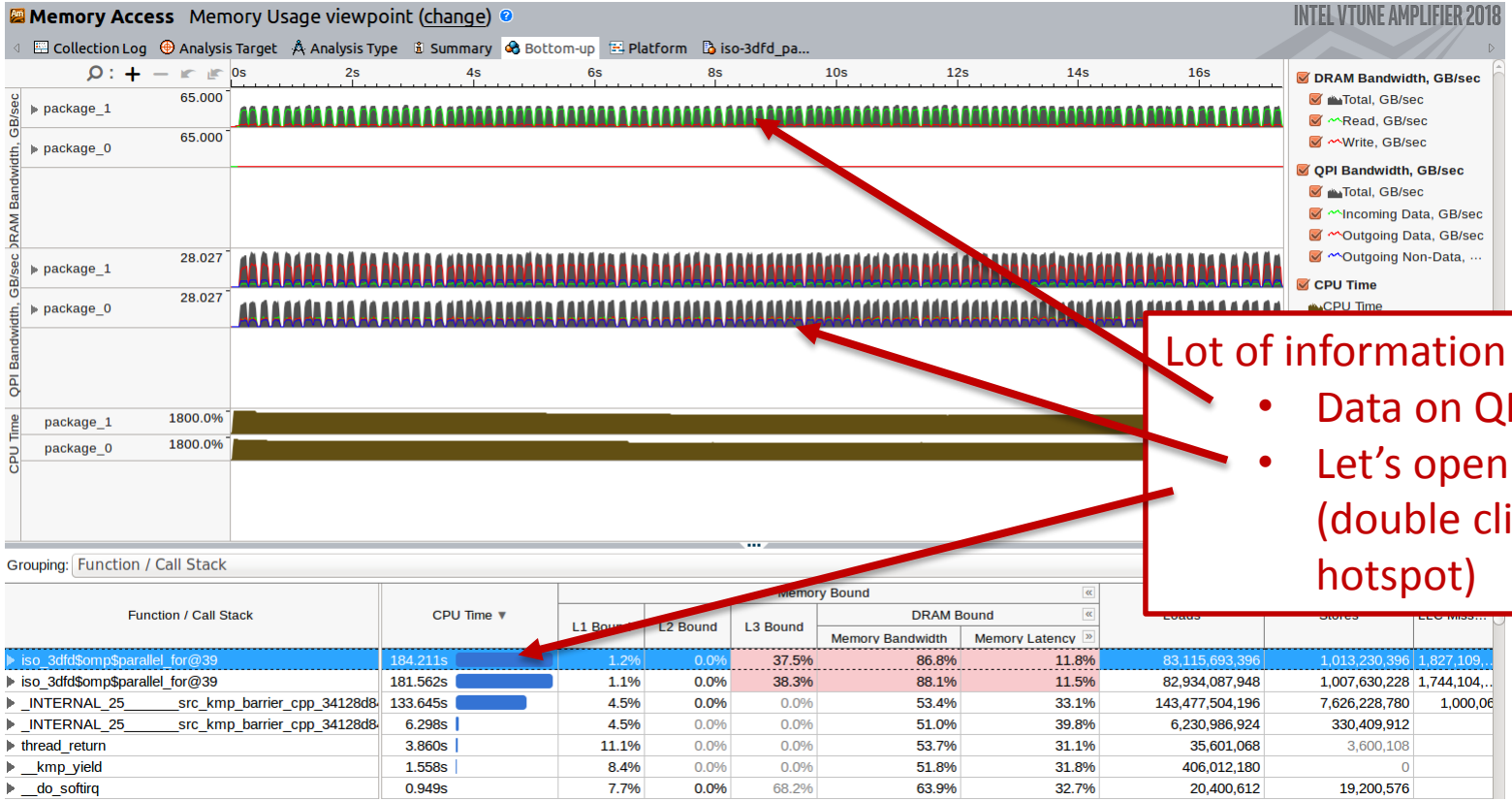
CPU Time:	515.074s	
Memory Bound:	56.2%	of Pipeline Slots
L1 Bound:	2.2%	of Clockticks
L2 Bound:	0.0%	of Clockticks
L3 Bound:	27.0%	of Clockticks
DRAM Bound:	27.1%	of Clockticks
DRAM Bandwidth Bound:	0.0%	of Elapsed Time
Memory Latency:		
Remote / Local memory Ratio:	0.661	
Loads:	327,307,818,940	
Stores:	17,299,718,976	
LLC Miss Count:	3,583,414,992	
Local DRAM Access Count:	1,592,847,784	
Remote DRAM Access Count:	1,052,431,572	
Remote Cache Access Count:	924,027,720	
Average Latency (cycles):	31	
Total Thread Count:	36	
Paused Time:	0s	



Lets look at the bottom up view and sort it by CPU Time

intel Software

BOTTOM UP VIEW OF MEMORY ACCESS ANALYSIS

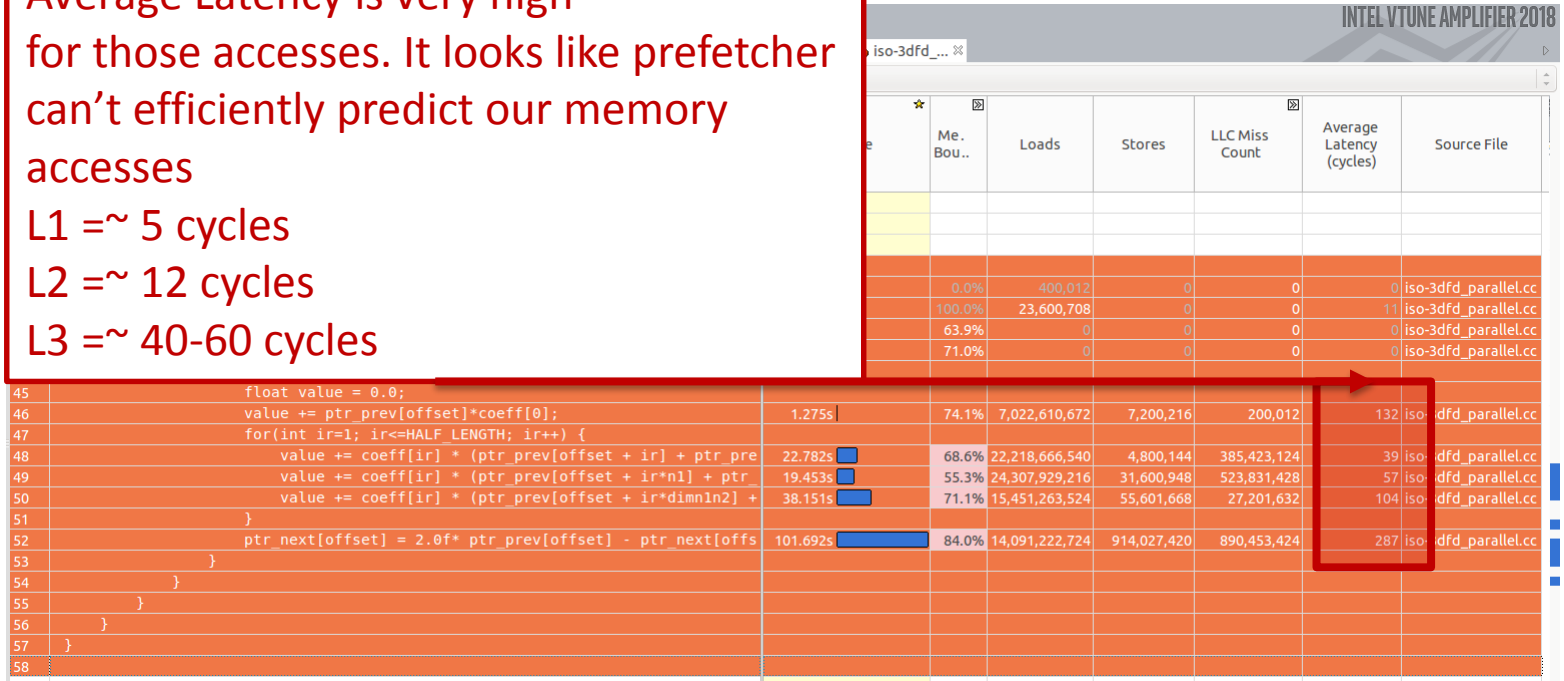


Lot of information

- Data on QPI
- Let's open the source (double clic on the main hotspot)

LET'S SEE THE SOURCE REPORT

Average Latency is very high
for those accesses. It looks like prefetcher can't efficiently predict our memory accesses
L1 =~ 5 cycles
L2 =~ 12 cycles
L3 =~ 40-60 cycles



PROBLEM CAN BE FOUND WITH ADVISOR

Summary | Survey & Roofline | Refinement Reports

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Max. Site Footprint
[loop in iso_3dfd at iso-3dfd_parallel.cc:..	No information available	100% / 0% / 0%	All unit strides	8B
[loop in iso_3dfd at iso-3dfd_parallel.cc:..	No information available	100% / 0% / 0%	All unit strides	65MB
[loop in iso_3dfd at iso-3dfd_parallel.cc:..	No information available	60% / 40% / 0%	Mixed strides	65MB
[loop in iso_3dfd at iso-3dfd_parallel.cc:..	No information available	60% / 40% / 0%	Mixed strides	65MB
[loop in iso_3dfd at iso-3dfd_parallel.cc:..	No information available	45% / 55% / 0%	Mixed strides	65MB
[loop in iso_3dfd at iso-3dfd_parallel.cc:..	No information available	45% / 55% / 0%	Mixed strides	65MB

Memory Access Patterns Report | Dependencies Report | Recommendations

ID	Stride	Type	Source	Nested Function	Variable references	Max. Site Footprint
P3	76800	Constant stride	iso-3dfd_parallel.cc:46		block 0x7fb15681b010 allocated at iso-3dfd_main.cc:183	65MB
P7	76800	Constant stride	iso-3dfd_parallel.cc:48		block 0x7fb15681b010 allocated at iso-3dfd_main.cc:183	65MB
P11	76800	Constant stride	iso-3dfd_parallel.cc:49		block 0x7fb15681b010 allocated at iso-3dfd_main.cc:183	65MB
P15	76800	Constant stride	iso-3dfd_parallel.cc:50		block 0x7fb15681b010 allocated at iso-3dfd_main.cc:183	65MB
P19	76800	Constant stride	iso-3dfd_parallel.cc:52		block 0x7fb14b851010 allocated at iso-3dfd_main.cc:185, block 0x7fb151036010 allocated at iso-3dfd_main.cc:184	65MB
P23	76800	Constant stride	iso-3dfd_parallel.cc:52		block 0x7fb151036010 allocated at iso-3dfd_main.cc:184	65MB
P29		Parallel site information	iso-3dfd_parallel.cc:43			
P56	0	Uniform stride	iso-3dfd_parallel.cc:46			4B
P65	0	Uniform stride	iso-3dfd_parallel.cc:48			4B
P71	0	Uniform stride	iso-3dfd_parallel.cc:49			8B
P81	0	Uniform stride	iso-3dfd_parallel.cc:50			8B
P91	0	Uniform stride	iso-3dfd_parallel.cc:52			8B

- Advisor reports many locations where we have a constant stride of 76800 happens (256*300=76800 which is the size of our 2 first dimensions)
- Constant strides are lines 46,48,49, 50,52

WHERE DOES THE CONSTANT STRIDE HAPPEN ?

```
void iso_3dfd_it(float *ptr_next, float *ptr_prev, float *ptr_vel, float
*coeff, const int n1, const int n2, const int n3, const int
num_threads, const int n1_Tblock, const int n2_Tblock, const int
n3_Tblock){
    int dimn1n2 = n1*n2;//This value will be used later
    #pragma omp parallel for default(shared)
    for(int ix=0; ix<n1; ix++) {
        for(int iy=0; iy<n2; iy++) {
            for(int iz=0; iz<n3; iz++) {
                // Constant stride appears here in the innermost loop
                int offset = iz*dimn1n2 + iy*n1 + ix;
                float value = 0.0;
                value += ptr_prev[offset]*coeff[0];
                …
            }
        }
    }
}
```

Iterations are not in the good order !!!

ISO3DFD DEV02 – RE-ORDERING THE LOOPS

CHANGING LOOP ORDER SOLVES THE PROBLEM

```
void iso_3dfd_it(float *ptr_next, float *ptr_prev, float *ptr_vel, float
*coeff, const int n1, const int n2, const int n3, const int
num_threads, const int n1_Tblock, const int n2_Tblock, const int
n3_Tblock){
    int dimn1n2 = n1*n2;//This value will be used later
    #pragma omp parallel for default(shared)
    for(int iy=0; iy<n2; iy++) {
        for(int iz=0; iz<n3; iz++) {
            for(int ix=0; ix<n1; ix++) {
                // Constant stride appears here in the innermost loop
                int offset = iz*dimn1n2 + iy*n1 + ix;
                float value = 0.0;
                value += ptr_prev[offset]*coeff[0];
                …
            }
        }
    }
}
```

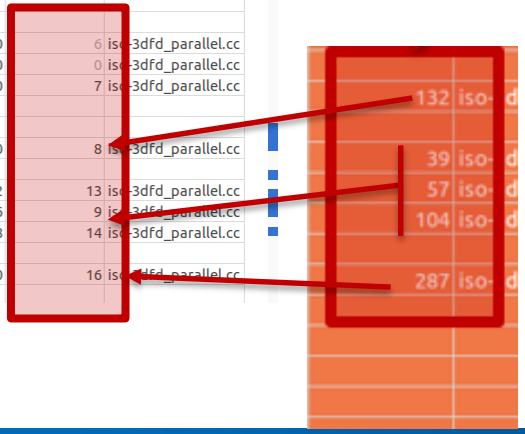
DID WE SOLVE THE LATENCY ISSUES ?

Memory Access Memory Usage viewpoint (change) — INTEL VTUNE AMPLIFIER 2018

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | Platform | iso-3dfd_...

Source | Assembly | Assembly grouping: Address

S Li.	Source	CPU Time	M B.	Loads	Stores	LLC Miss Count	Average Latency (cycles)	Source File
32	#include <omp.h>							
33	#include <stdio.h>							
34								
35	void iso_3dfd_it(float *ptr_next, float *ptr_prev, float *ptr_vel, float							
36	const int n1, const int n2, const int n3, const int num_threads,							
37	const int n1_Tblock, const int n2_Tblock, const int n3_Tblock){							
38	int dimn1n2 = n1*n2;//This value will be used later							
39	#pragma omp parallel for default(shared)							
40	for(int iz=0; iz<n3; iz++) {							
41	for(int iy=0; iy<n2; iy++) {	0.001s		0	0	0	6	iso-3dfd_parallel.cc
42	for(int ix=0; ix<n1; ix++) {	0.062s	0.0%	0	0	0	0	iso-3dfd_parallel.cc
43	if(ix>=HALF_LENGTH && ix<(n1-HALF_LENGTH) && iy>=HALF_LENGTH	0.640s	1.4%	116,803,504	400,012	0	7	iso-3dfd_parallel.cc
44	int offset = iz*dimn1n2 + iy*n1 + ix;							
45	float value = 0.0;							
46	value += ptr_prev[offset]*coeff[0];	0.311s	48.2..	7,200,216	400,012	0	8	iso-3dfd_parallel.cc
47	for(int ir=1; ir<=HALF_LENGTH; ir++) {							
48	value += coeff[ir] * (ptr_prev[offset + ir] + ptr_pre	24.549s	50.8..	32,355,770,644	10,400,312	200,012	13	iso-3dfd_parallel.cc
49	value += coeff[ir] * (ptr_prev[offset + ir*n1] + ptr_	9.205s	46.5..	7,477,824,328	40,401,212	2,600,156	9	iso-3dfd_parallel.cc
50	value += coeff[ir] * (ptr_prev[offset + ir*dimn1n2] +	9.486s	33.9..	21,197,035,892	52,001,560	23,801,428	14	iso-3dfd_parallel.cc
51	}							
52	ptr_next[offset] = 2.0f* ptr_prev[offset] - ptr_next[offs	3.186s	34.0..	8,157,844,728	793,623,808	7,000,420	16	iso-3dfd_parallel.cc
53	}							

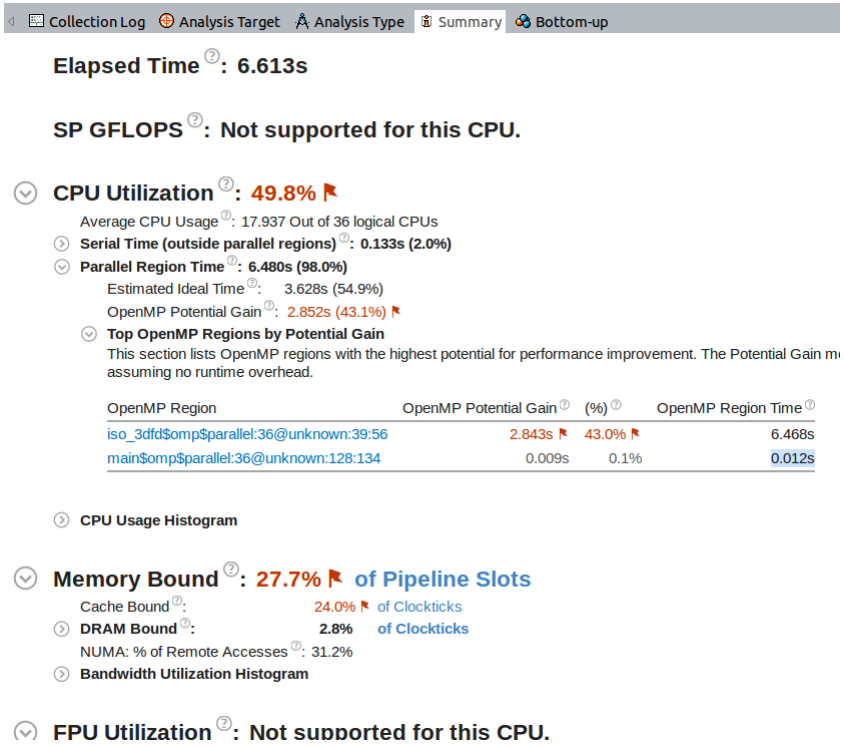


intel Software

WHAT IS COMING NEXT ?

- We enter the iterating process again
  - Run **HPC Performance Characterization**
  - Detect the strongest bottleneck
  - New: Use **Advisor** if not enough information available
- Remember that **Vtune** is perfect for understanding
  - Threading issues
  - Memory issues
- **Advisor** can help you to
  - Optimize vectorization
  - Understand how far you are from the maximum performance
  - Investigate data access

HPC PERFORMANCE CHARACTERIZATION ON DEVO2



- We went from 15.2s to 6.6s (=~2.3X additional speedup)
- Threading seems a bit low but the test case might be a bit small
- Let's try to increase the size of data to see if something shows up

ISO3DFD DEV03 – CACHE BLOCKING

MEMORY ISSUES ?

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up | iso-3dfd_pa...

Elapsed Time: 31.720s

SP GFLOPS: Not supported for this CPU.

CPU Utilization: 61.8%

Average CPU Usage: 22.237 Out of 36 logical CPUs

Serial Time (outside parallel regions): 0.517s (1.6%)

Top Serial Hotspots (outside parallel regions)

Parallel Region Time: 31.203s (98.4%)

Estimated Ideal Time: 27.163s (85.6%)

OpenMP Potential Gain: 4.040s (12.7%)

Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that
runtime overhead.

OpenMP Region	OpenMP Potential Gain	(%)	OpenMP Region Time
iso_3dfdompparallel:36@unknown:39:56	4.035s	12.7%	31.197s
mainompparallel:36@unknown:128:134	0.005s	0.0%	0.006s

CPU Usage Histogram

Memory Bound: 49.9% of Pipeline Slots

Cache Bound: 45.1% of Clockticks

DRAM Bound: 4.6% of Clockticks

DRAM Bandwidth Bound: 0.0% of Elapsed Time

NUMA: % of Remote Accesses: 56.8%

Bandwidth Utilization Histogram

Explore bandwidth utilization over time using the histogram and identify memory objects functions with maximum contribution to the high bandwidth

Bandwidth Domain: DRAM, GB/sec

Bandwidth Utilization Histogram

Top Functions with High Bandwidth Utilization

Looking at the bottom up analysis might give more details

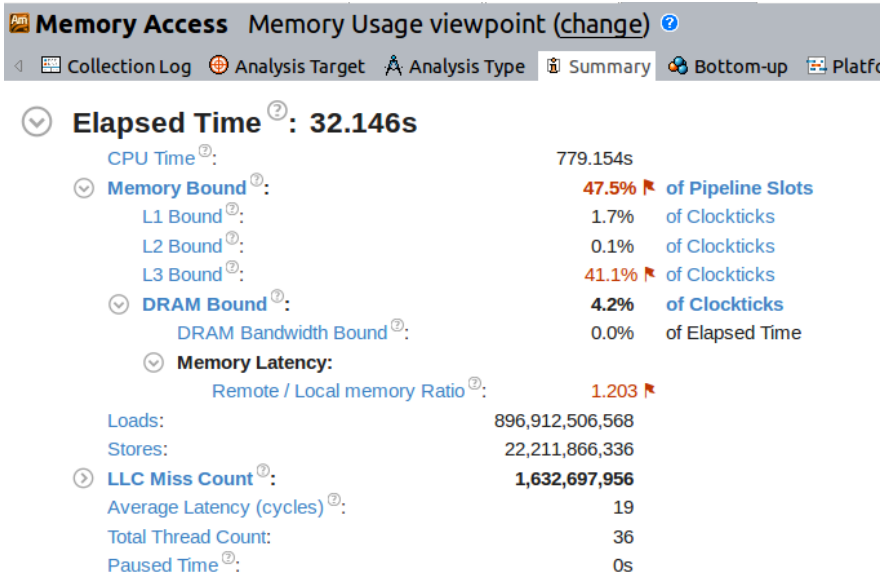
HPC PERFORMANCE CHARACTERIZATION INDICATES MEMORY BOUND EXECUTION

We might have memory issues, Let's use the memory access analysis to investigate.

S			Memory Bound	Source File
31	#include "iso-3dfd.h			
32	#include <omp.h>			
33	#include <stdio.h>			
34				
35	void iso_3dfd_it(flo			
36	const int			
37	const int			
38	int dimn1n2 = n1*n2;//this value will be used later			
39	#pragma omp parallel for default(shared)			
40	for(int iz=0; iz<n3; iz++) {			
41	for(int iy=0; iy<n2; iy++) {			
42	for(int ix=0; ix<n1; ix++) {	0.191s	0.0%	iso-3dfd_parallel.cc
43	if(ix>=HALF_LENGTH && ix<(n1-HALF_LENGTH) && iy>=HALF_LENGTH && i	3.291s	9.8%	iso-3dfd_parallel.cc
44	int offset = iz*dimn1n2 + iy*n1 + ix;			
45	float value = 0.0;			
46	value += ptr_prev[offset]*coeff[0];	1.867s	36.0%	iso-3dfd_parallel.cc
47	for(int ir=1; ir<=HALF_LENGTH; ir++) {			
48	value += coeff[ir] * (ptr_prev[offset + ir] + ptr_prev[off	179.820s	61.2%	iso-3dfd_parallel.cc
49	value += coeff[ir] * (ptr_prev[offset + ir*n1] + ptr_prev[	73.237s	60.1%	iso-3dfd_parallel.cc
50	value += coeff[ir] * (ptr_prev[offset + ir*dimn1n2] + ptr_	74.715s	48.6%	iso-3dfd_parallel.cc
51	}			
52	ptr_next[offset] = 2.0f* ptr_prev[offset] - ptr_next[offset] +	20.890s	38.9%	iso-3dfd_parallel.cc
53	}			
54	}			
55	}			

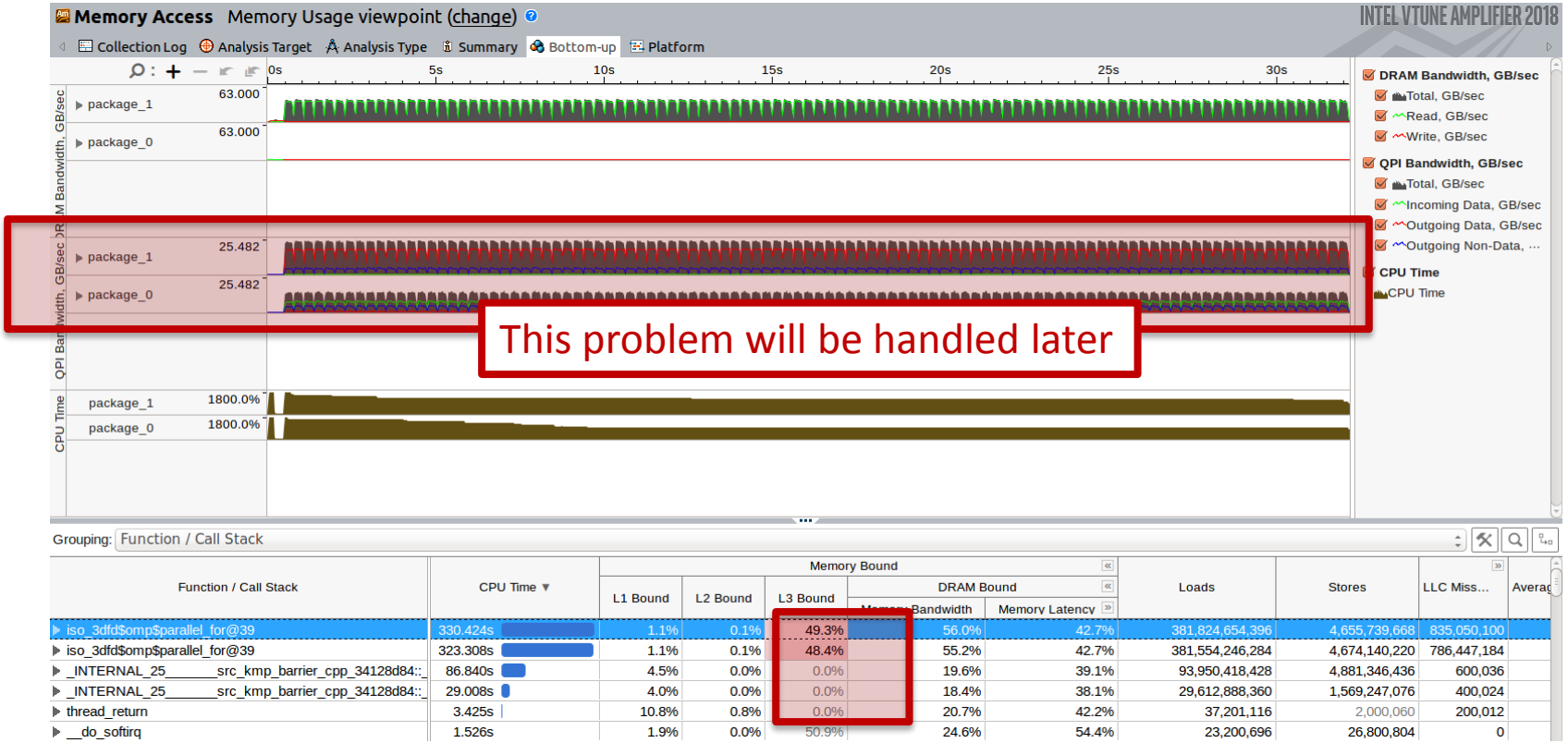
L3 BOUND ?

- It looks like we are spending a significant amount of time waiting data from the L3



In a ideal case, we would expect to be L1 bound (meaning that data mainly come from L1)
The bottom up view can give more precise information

MEMORY ACCESS ANALYSIS – BOTTOM UP VIEW



Function / Call Stack	CPU Time ▼	Memory Bound					Loads	Stores	LLC Miss...	Averag
		L1 Bound	L2 Bound	L3 Bound	DRAM Bound					
					Memory Bandwidth	Memory Latency				
iso_3dfdompparallel_for@39	330.424s	1.1%	0.1%	49.3%	56.0%	42.7%	381,824,654,396	4,655,739,668	835,050,100	
iso_3dfdompparallel_for@39	323.308s	1.1%	0.1%	48.4%	55.2%	42.7%	381,554,246,284	4,674,140,220	786,447,184	
_INTERNAL_25_____src_kmp_barrier_cpp_34128d84::_	86.840s	4.5%	0.0%	0.0%	19.6%	39.1%	93,950,418,428	4,881,346,436	600,036	
_INTERNAL_25_____src_kmp_barrier_cpp_34128d84::_	29.008s	4.0%	0.0%	0.0%	18.4%	38.1%	29,612,888,360	1,569,247,076	400,024	
thread_return	3.425s	10.8%	0.8%	0.0%	20.7%	42.2%	37,201,116	2,000,060	200,012	
__do_softirq	1.526s	1.9%	0.0%	50.9%	24.6%	54.4%	23,200,696	26,800,804	0	

LATENCY IS STILL IMPORTANT ON A BIGGER TESTCASE

Remainder:
L1 =~ 5 cycles
L2 =~ 12 cycles
L3 =~ 40-60 cycles
So the number of cycles is still high, 20-40 cycles means that many accesses are resolved in L3.

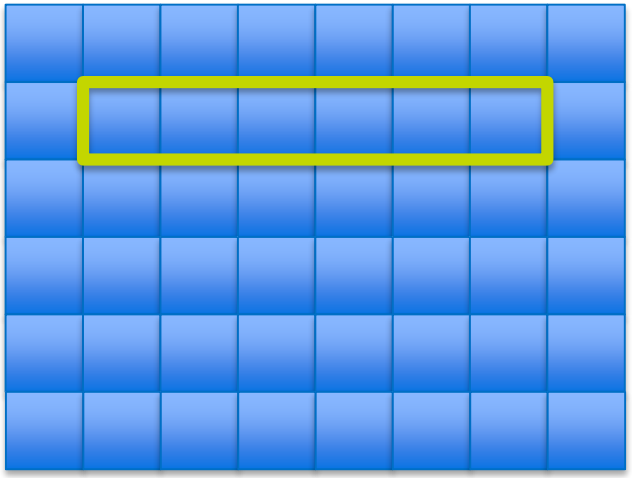
S Li.	Source	CPU Time						
32	#include <omp.h>							
33	#include <stdio.h>							
34								
35	void iso_3dfd_it(float *ptr_next, float *ptr_prev, float *ptr_vel, float							
36	const int n1, const int n2, const int n3, const int num_threads,							
37	const int n1_Tblock, const int n2_Tblock, const int n3_Tblock){							
38	int dimn1n2 = n1*n2;//This value will be used later							
39	#pragma omp parallel for default(shared)							
40	for(int iz=0; iz<n3; iz++) {							
41	for(int iy=0; iy<n2; iy++) {	0.011s	17.9%	0	0	0	33	iso-3dfd_parallel.cc
42	for(int ix=0; ix<n1; ix++) {	0.233s	0.0%	0	0	0	0	iso-3dfd_parallel.cc
43	if(ix>=HALF_LENGTH && ix<(n1-HALF_LENGTH) && iy>=HALF_LENGTH	3.198s	9.4%	1,106,833,204	800,024	0	7	iso-3dfd_parallel.cc
44	int offset = iz*dimn1n2 + iy*n1 + ix;							
45	float value = 0.0;							
46	value += ptr_prev[offset]*coeff[0];	1.852s	36.1%	102,003,060	800,024	3,400,204	12	iso-3dfd_parallel.cc
47	for(int ir=1; ir<=HALF_LENGTH; ir++) {							
48	value += coeff[ir] * (ptr_prev[offset + ir] + ptr_pre	168.898s	61.3%	221,545,846,...	56,001,680	22,201,332	19	iso-3dfd_parallel.cc
49	value += coeff[ir] * (ptr_prev[offset + ir*n1] + ptr_	67.650s	57.9%	39,676,790,268	92,002,760	17,801,068	13	iso-3dfd_parallel.cc
50	value += coeff[ir] * (ptr_prev[offset + ir*dimn1n2] +	69.015s	46.2%	86,685,800,496	160,804,824	749,644,976	39	iso-3dfd_parallel.cc
51	}							
52	ptr_next[offset] = 2.0f* ptr_prev[offset] - ptr_next[offs	19.567s	39.7%	32,707,381,192	4,345,330,356	42,002,520	23	iso-3dfd_parallel.cc
53	}							

CACHE BLOCKING

- We observed that our data mainly come from L3
  - Average number of cycle for accessing data is high (close to L3 latency)
  - Vtune reports the kernel to be memory bound in L3
- We can try to implement Cache Blocking
  - Cache blocking consists in creating blocks in the 3D volume and to compute block by block
  - It increases the chance to benefit from data locality

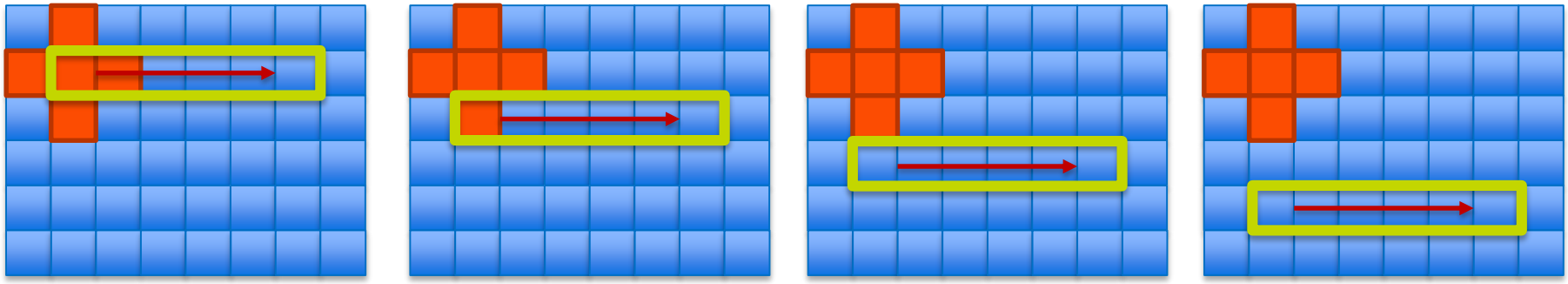
STENCIL WITHOUT CACHE BLOCKING

- We refresh cells following unit stride order



STENCIL WITHOUT CACHE BLOCKING

- We refresh cells following unit stride order

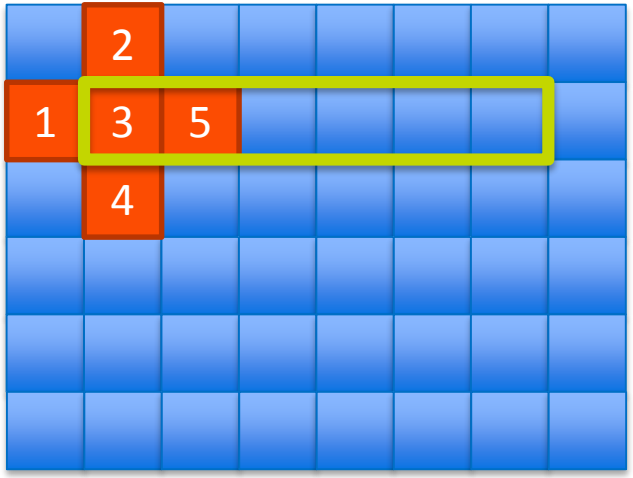


- We iterate with 2 for loops

```
for(int i=0;i<imax;i++)
  for(int j=0; j<jmax; j++)
    tab[i][j]=...
```

STENCIL WITHOUT CACHE BLOCKING

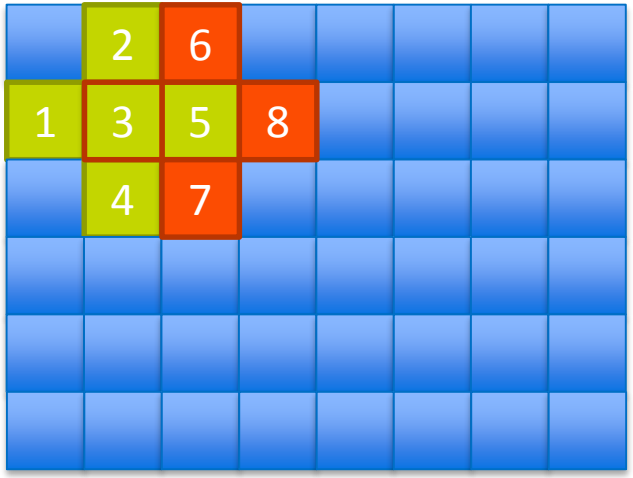
- Let's assume we can store 16 cells in cache (fifo policy)



0 cache evictions
5 cache misses

STENCIL WITHOUT CACHE BLOCKING

- Let's assume we can store 16 cells in cache (fifo policy)

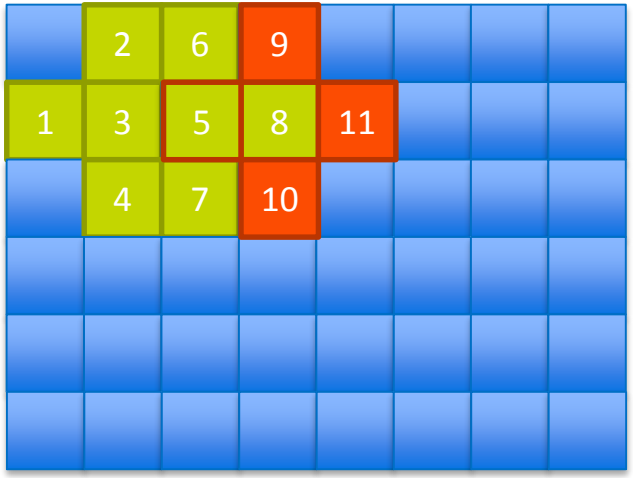


Memory cell still in cache

0 cache evictions
8 cache misses

STENCIL WITHOUT CACHE BLOCKING

- Let's assume we can store 16 cells in cache (fifo policy)



Memory cell still in cache

0 cache evictions
11 cache misses

STENCIL WITHOUT CACHE BLOCKING

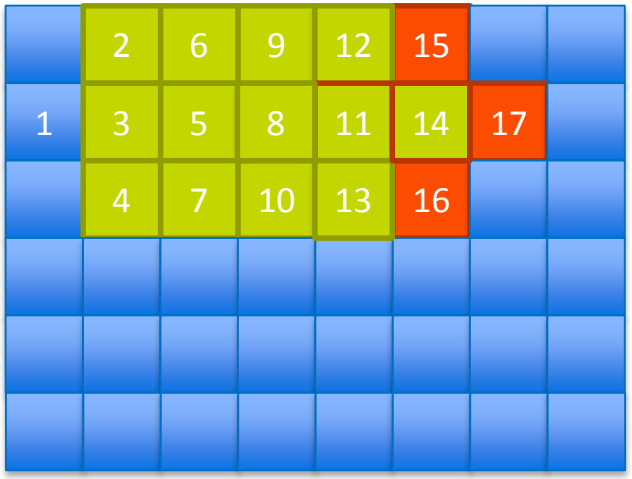
- Let's assume we can store 16 cells in cache (fifo policy)

Memory cell still in cache

0 cache evictions
14 cache misses

STENCIL WITHOUT CACHE BLOCKING

- Let's assume we can store 16 cells in cache (fifo policy)



Memory cell still in cache

1 cache evictions
17 cache misses

STENCIL WITHOUT CACHE BLOCKING

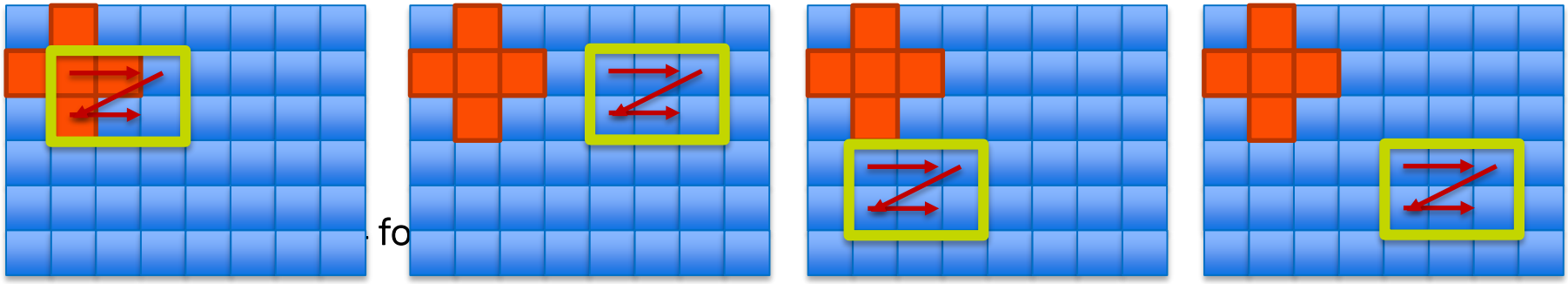
- Let's assume we can store 16 cells in cache (fifo policy)

Memory cell still in cache

4 cache evictions
20 cache misses

STENCIL WITH CACHE BLOCKING

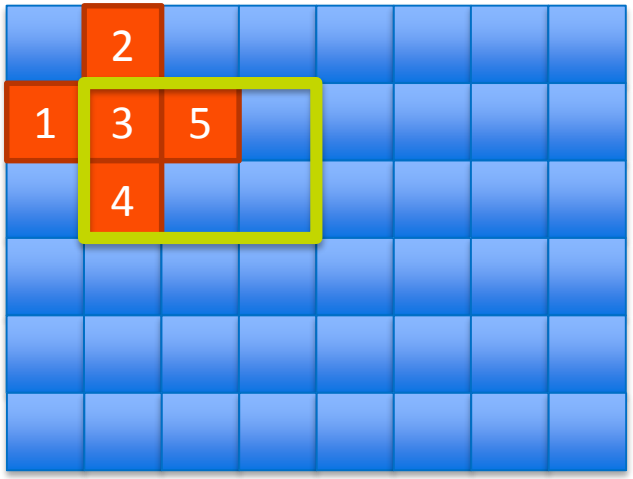
- We create blocks and we iterate using unit strides inside the blocks.



```
for(int bi=0;bi<imax;i+=BI)
  for(int bj=0; bj<jmax; j+=BJ)
    for(int i=bi; i<bi+BI; i++)
      for(int j=bj; j<bj+BJ; j++)
        tab[i][j]=...
```

STENCIL WITH CACHE BLOCKING

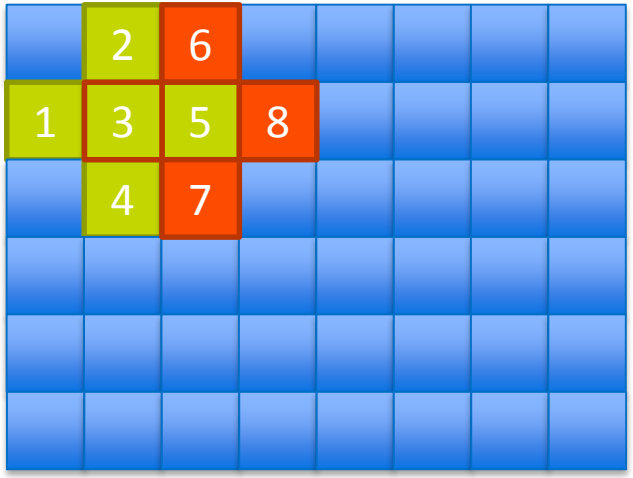
- Let's assume we can store 16 cells in cache (fifo policy)



0 cache evictions
5 cache misses

STENCIL WITHOUT CACHE BLOCKING

- Let's assume we can store 16 cells in cache (fifo policy)

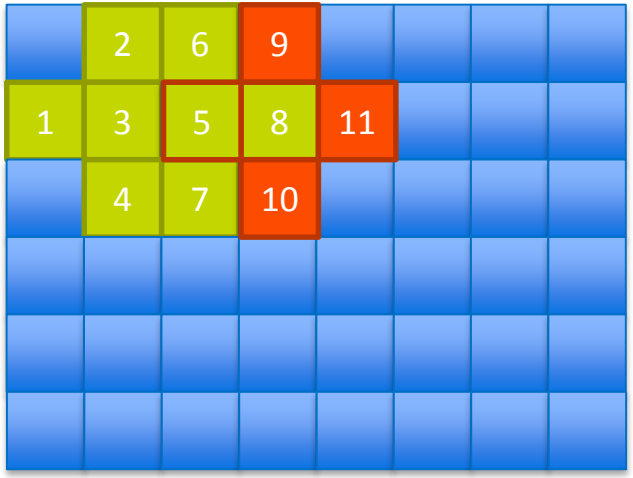


Memory cell still in cache

0 cache evictions
8 cache misses

STENCIL WITHOUT CACHE BLOCKING

- Let's assume we can store 16 cells in cache (fifo policy)

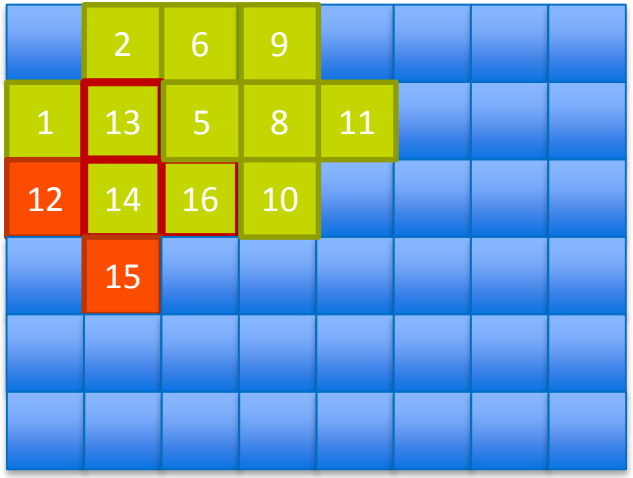


Memory cell still in cache

0 cache evictions
11 cache misses

STENCIL WITHOUT CACHE BLOCKING

- Let's assume we can store 16 cells in cache (fifo policy)

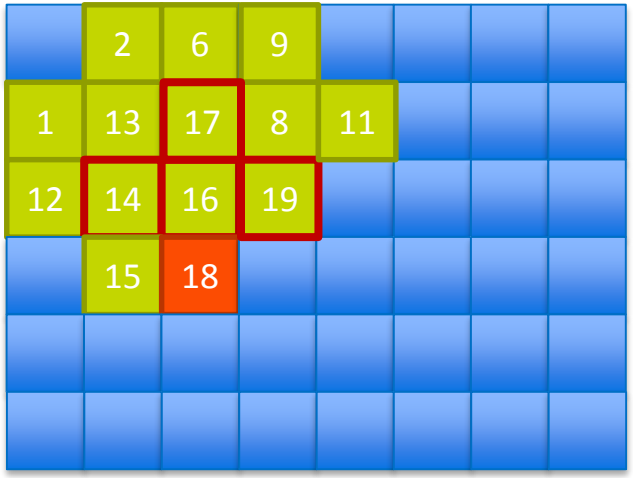


Memory cell still in cache

0 cache evictions
13 cache misses

STENCIL WITHOUT CACHE BLOCKING

- Let's assume we can store 16 cells in cache (fifo policy)

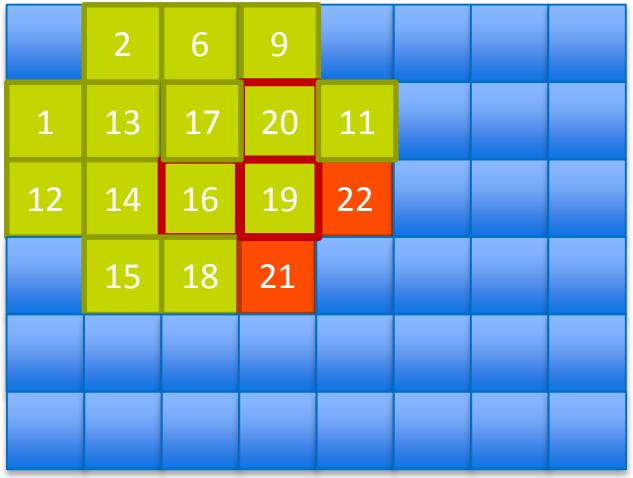


Memory cell still in cache

0 cache evictions
14 cache misses

STENCIL WITHOUT CACHE BLOCKING

- Let's assume we can store 16 cells in cache (fifo policy)



Memory cell still in cache

0 cache evictions
16 cache misses

COMPARISON ?

- In both cases we update 6 elements (center of the stencil)
- Without cache blocking
  - 4 cache eviction
  - 20 cache misses
- With cache blocking
  - 0 cache eviction
  - 16 cache misses

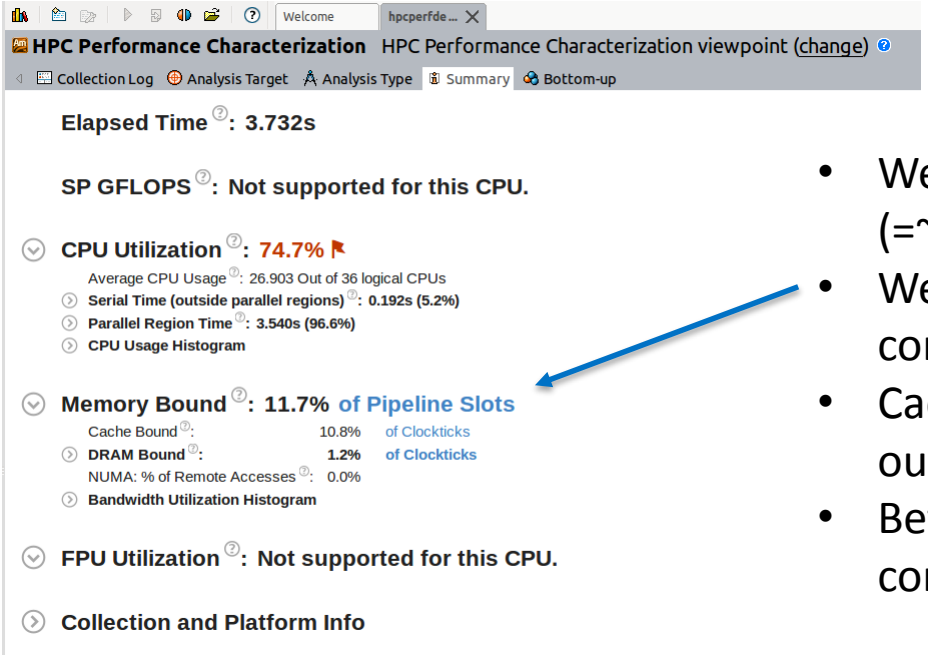
HOW TO IMPLEMENT CACHE BLOCKING ?

- Need to create an additional for loop for each dimension
  - General principle
  - In practice, you usually don't want to do cache blocking in the unit stride dimension (it breaks prefetching)
  - It adds a significant overhead in code readability
- Other solution, use TBB if you are in C++
  - Use blocked_range feature of TBB

BASIC IMPLEMENTATION

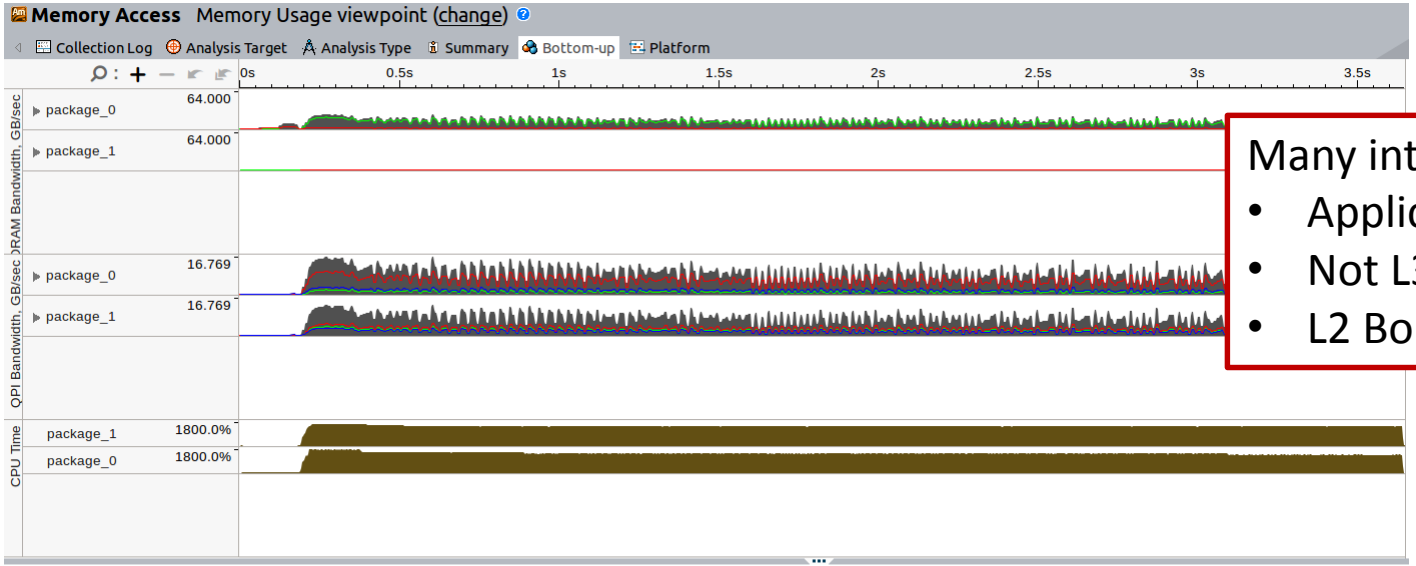
```
int nZEnd = nZ - HALF_LENGTH;
int nYEnd = nY - HALF_LENGTH;
int nXEnd = nX - HALF_LENGTH;

#pragma omp parallel for OMP_SCHEDULE OMP_N_THREADS collapse(3) default(shared)
for(int bz=HALF_LENGTH; bz<nZEnd; bz+=nZ_Sblock){
 for(int by=HALF_LENGTH; by<nYEnd; by+=nY_Sblock){
  for(int  bx=HALF_LENGTH; bx<nXEnd; bx+=nX_Sblock){
   int izEnd = MIN(bz+nZ_Sblock, nZEnd);
   int iyEnd = MIN(by+nY_Sblock, nYEnd);
   int ixEnd = MIN(bx+nX_Sblock, nXEnd);
   for(int iz=bz; iz<izEnd; iz++) {
    for(int iy=by; iy<iyEnd; iy++) {
     for(int ix=bx; ix<ixEnd; ix++) {
```

WHAT IS THE PERFORMANCE OF DEV03 ?



- We went from 6.6s to 3.7s (=~1.7X additional speedup)
- We are back to the small test case to have fair comparison
- Cache blocking seems to significantly improve our memory accesses
- Better CPU utilization due to less memory contention

WHAT ABOUT THE L3 AND L2 NOW ?



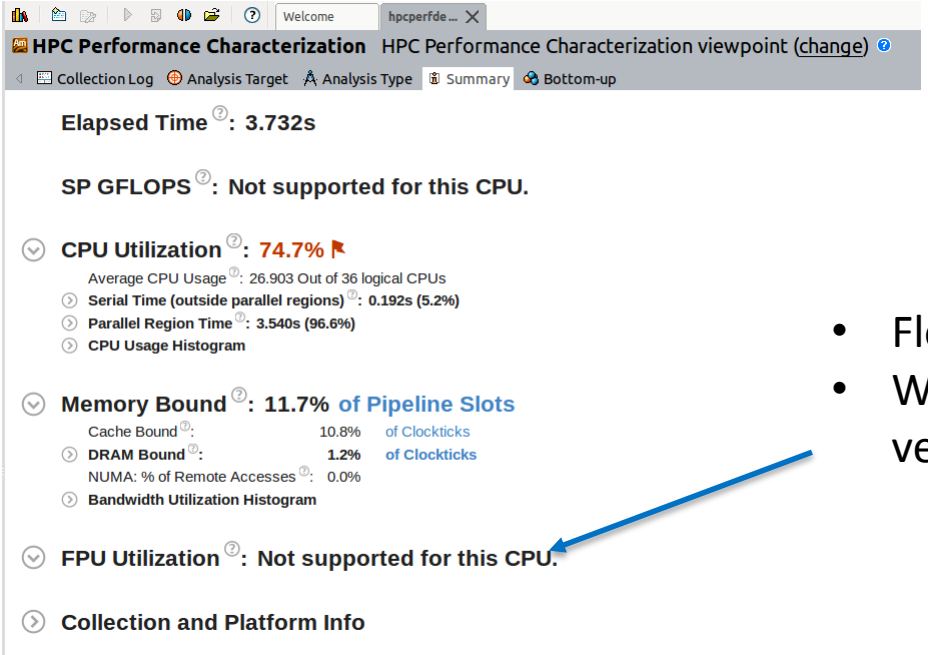
Grouping: Function / Call Stack

Function / Call Stack	CPU Time ▼	Memory Bound					Loads	Stor
		L1 Bound	L2 Bound	L3 Bound	DRAM Bound			
					Memory Bandwidth	Memory Latency		
iso_3dfdompparallel_for@51	48.559s	1.7%	7.1%	2.5%	11.9%	72.8%	72,316,969,444	1,12
iso_3dfdompparallel_for@51	48.275s	1.6%	7.2%	2.3%	11.4%	70.0%	72,219,366,516	1,13
_INTERNAL_25_____src_kmp_barrier_cpp_34128d84::_	4.849s	4.7%	0.0%	0.0%	47.1%	36.9%	4,976,149,280	29
_INTERNAL_25_____src_kmp_barrier_cpp_34128d84::_	0.264s	3.2%	0.0%	1.1%	44.7%	36.2%	303,609,108	1
do_softirq	0.144s	10.5%	23.0%	14.7%	21.0%	62.9%	3,600,108	

Many interesting points:

- Application runs faster
- Not L3 bounded anymore
- L2 Bound > L3 Bound

TIME TO LOOK AT VECTORIZATION ?



- Flops counters not available on this hardware ?
- We can use Intel® Advisor to enable vectorization

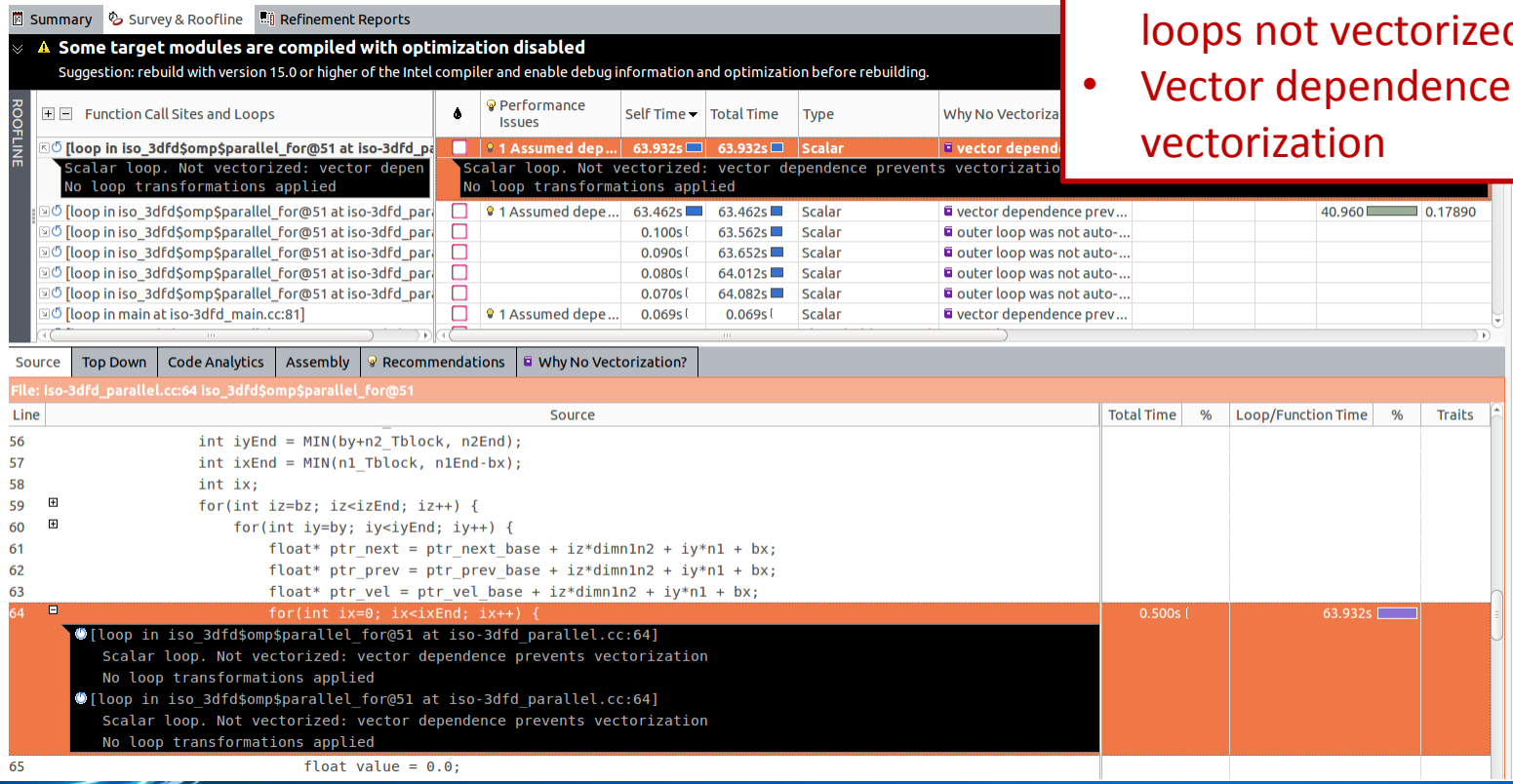
HOW CAN ADVISOR EXTRACT VECTORIZATION INFORMATIONS ?

- Vtune relies on Hardware counter to extract flops
  - No available hardware counter = No flops reported
  - Hardware counters have been broken on our previous platforms
  - Few ms run might result on non-accurate results
- Intel® Advisor uses instrumentation
  - Advisor can extract data from the binary (from internal markers that the compiler put in your executable)
  - Advisor looks for instructions executed and count **real flops** (takes masking into account)
  - It only uses sampling for the timings

ISO3DFD DEV04 – VECTORIZATION

USING INTEL® ADVISOR ON DEVO3

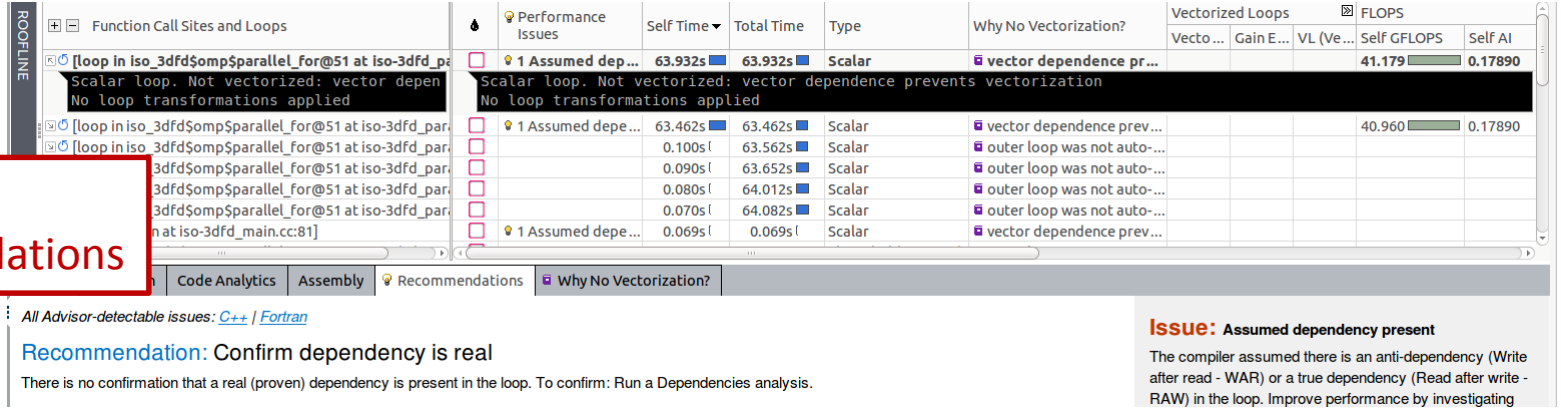
- Advisor detects 2 significant loops not vectorized
- Vector dependence prevents vectorization

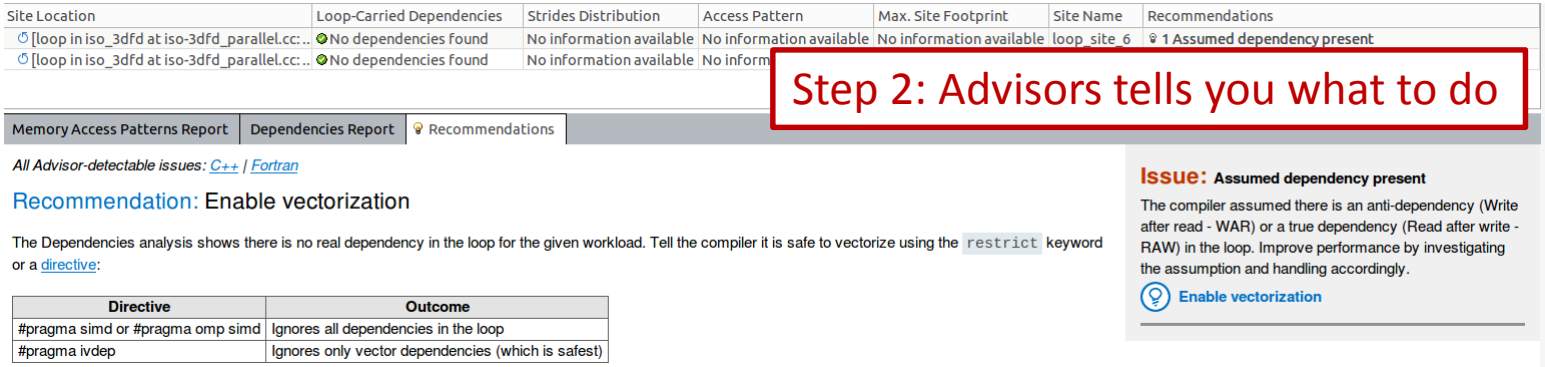


intel Software

DEPENDENCY ANALYSIS ON DEVCLOUD

Step 1: Look at recommendations



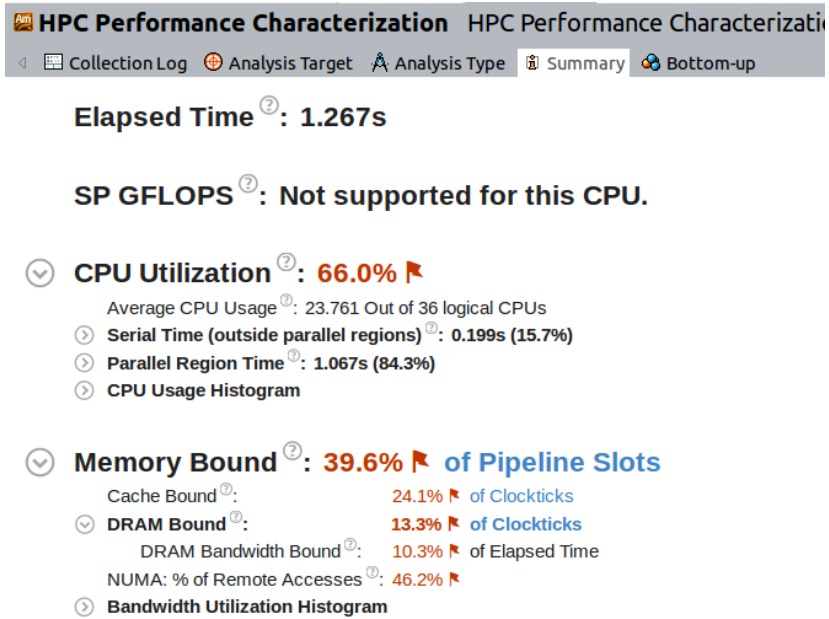


Step 2: Advisors tells you what to do

ADDING VECTORIZATION ON DEV03

```
#pragma omp simd
for(int ix=0; ix<ixEnd; ix++) {
  float value = 0.0;
  value += ptr_prev[ix]*coeff[0];
  #pragma unroll(HALF_LENGTH)
  for(int ir=1; ir<=HALF_LENGTH; ir++) {
    value += coeff[ir] * (ptr_prev[ix + ir] + ptr_prev[ix - ir]);// horizontal
    value += coeff[ir] * (ptr_prev[ix + ir*n1] + ptr_prev[ix - ir*n1]);// vertical
    value += coeff[ir] * (ptr_prev[ix + ir*dimn1n2] + ptr_prev[ix - ir*dimn1n2]); // in front / behind
  }
  ptr_next[ix] = 2.0f* ptr_prev[ix] - ptr_next[ix] + value*ptr_vel[ix];
```

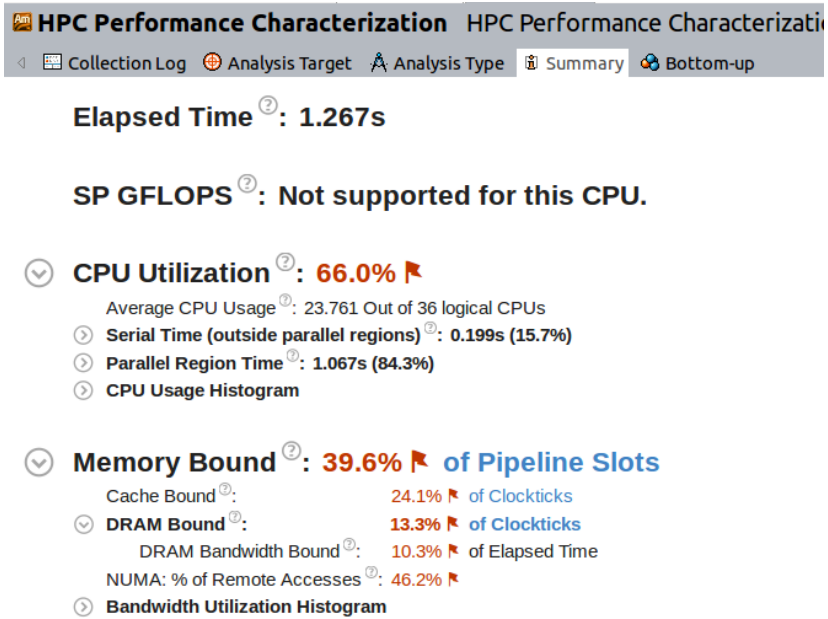
DEV04 – HPC PERFORMANCE CHARACTERIZATION



- We went from 3.7s to 1.2s (=~3X additional speedup)
- As we are doing more flops, we request also more data.

ISO3DFD DEV05 – NUMA EFFECT

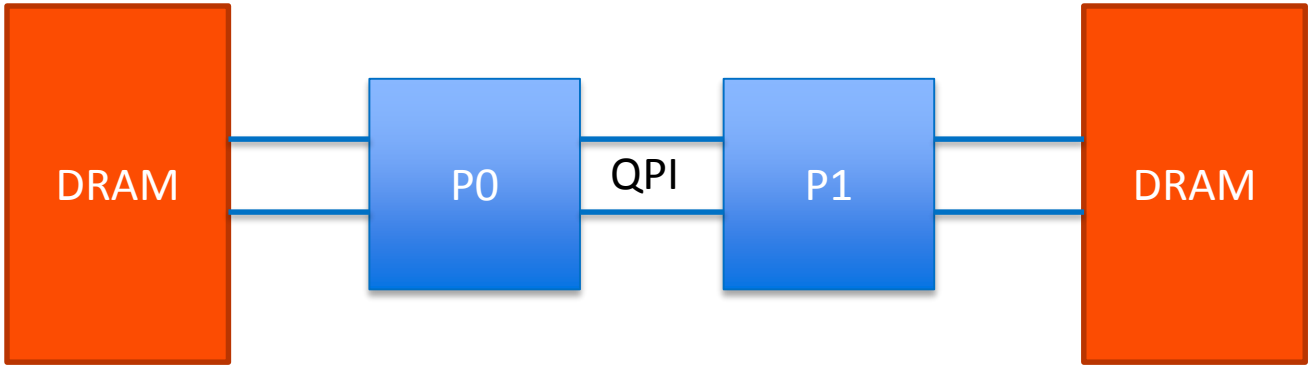
DEV04 – REMOTE MEMORY ACCESSES ?



- We have done significant progress
- But memory seems to be a bottleneck again
- Vtune indicates a high ratio of NUMA accesses

WHAT IS NON UNIFORM MEMORY ACCESS (NUMA)

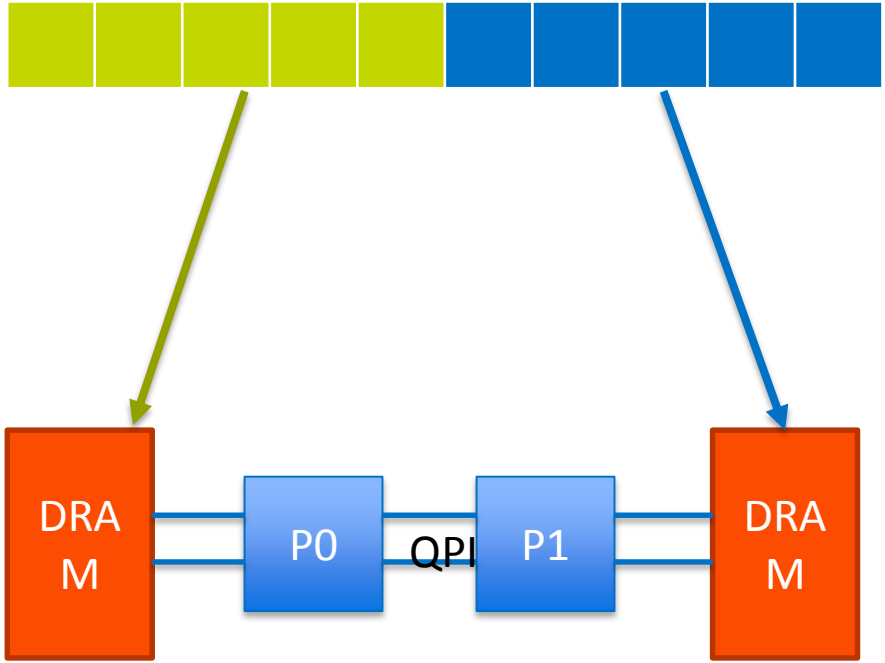
- **Wikipedia:** Non-uniform memory access (NUMA) is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor.
- Multi-socket system are sensitive to NUMA



HOW TO SOLVE NUMA PROBLEMS ?

- Implements First touch policy

```
#pragma omp parallel for schedule(static)
for(int i=0;i<N;i++){
  tab[i] = 0;
  output[i] = 0;
}
//initialize output and tab
real_init(tab, output);

#pragma omp parallel for schedule(static)
for(int i=0;i<N;i++)
  output[i] = tab[i]*.....
```



HOW TO SOLVE NUMA PROBLEMS 2 ?

- Use MPI
  - You need to modify your application to handle MPI domain decomposition
  - Use MPI runtime to bind one process per NUMA node (socket)
  - You have to handle communications between your nodes
  - If a process is bind to a socket, it will allocate the memory on the closest NUMA node.

HOW TO TEST YOUR IDEAL PERFORMANCE ?

- Run your application with a given test case
  - Look at the timing
  - You can also check bandwidth on QPI with Vtune
- Run 2 instances of your application with NUMA CTL
  - Divide the amount of work by 2
  - Bind 1 process on node 0 and assign memory allocation on socket 0
  - Bind 1 process on node 1 and and assign memory allocation on socket 1

HOW TO TEST YOUR IDEAL PERFORMANCE ?

- bin/iso3dfd_dev05_cpu_avx2.exe 256 300 300 36 100

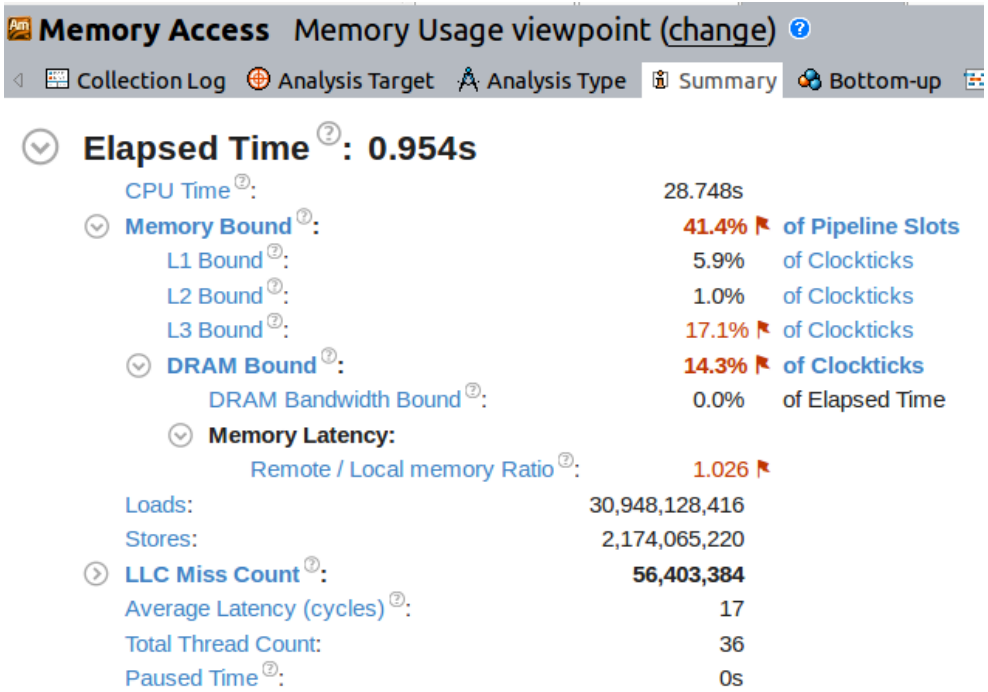
  real 0m1.348s
  user 0m39.340s
  sys 0m0.500s

- We went from 1.3s to 0.6s (=~2.16X additional speedup)

- numactl --cpunodebind=0 --membind=0 bin/iso3dfd_dev05_cpu_avx2.exe 128 300 300 18 100 &
  numactl --cpunodebind=1 --membind=1 bin/iso3dfd_dev05_cpu_avx2.exe 128 300 300 18 100

  real 0m0.558s
  user 0m9.296s
  sys 0m0.338s

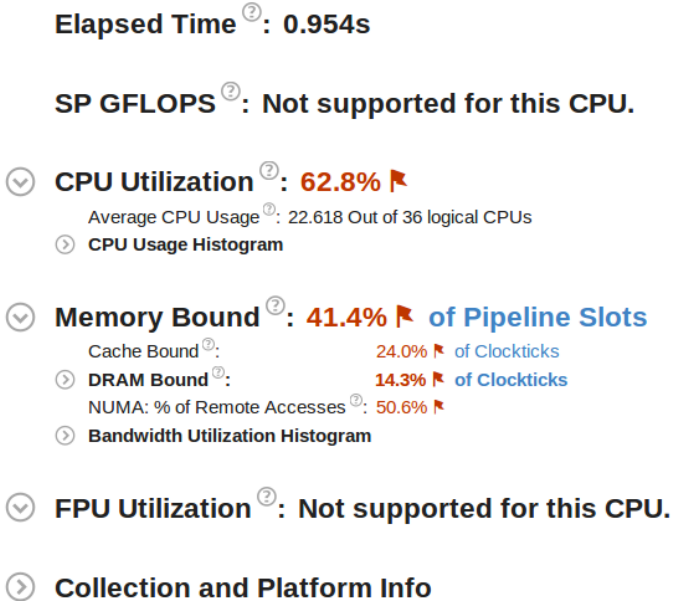
WITH FIRST TOUCH ?



- We went from 1.2s to 0.9s (=~1.3X additional speedup)
- Overall 315 to 0.9s with
  - Threading
  - Vectorization
  - Cache blocking
  - Memory optimizations
- We can get even more if we tune cache blocking parameters

FIRST TOUCH MEMORY ACCESS

- How to improve ?



We might want to specify OMP scheduling

- **It might improve threading**
- **It might improve NUMA accesses**

FIRST TOUCH MEMORY ACCESS

- Export KMP_AFFINITY=compact

Elapsed Time ⓘ: **0.754s**

SP GFLOPS ⓘ: **Not supported for this CPU.**

CPU Utilization ⓘ: **80.2%**

Average CPU Usage ⓘ: 28.876 Out of 36 logical CPUs

CPU Usage Histogram

Memory Bound ⓘ: **50.3%** of Pipeline Slots

Cache Bound ⓘ: 29.0% of Clockticks

DRAM Bound ⓘ: **18.0%** of Clockticks

NUMA: % of Remote Accesses ⓘ: 43.1%

Bandwidth Utilization Histogram

FPU Utilization ⓘ: **Not supported for this CPU.**

Collection and Platform Info

- We went from 0.9s to 0.7s (=~1.3X additional speedup)
- Overall 315 to 0.7s

CONCLUSION

- Use Vtune for:
  - Threading (HPC PERFORMANCE CHARACTERIZATION)
  - Memory optimization (MEMORY ACCESS ANALYSIS)
    - Look for Cache Bound, Memory bound, Cache latency
    - Look for data on QPI
- Use Advisor for:
  - Vectorization
  - Profiling data accesses
  - Characterization (Roofline model – not covered here)


intel
Software