A Relationship Guide For Your Hardware

 $\bullet \bullet \bullet$

How to understand the needs of your hardware: Introduction to data based low-level optimization

Felix Klinge - Senior Software Engineer

Agenda

- About me
- High Level vs Low Level optimizations
- Why should I care about low level stuff?
- Example: 2D bitmap rotation
- Naive Implementation
- Optimization 1: Better Execution Unit Utilization
- Optimization 2: Loop Blocking
- Optimization 3: Multithreading
- Optimization 4: SIMD
- Conclusion

About me

- In the games industry for ~10 years
- Worked mostly on custom engine/game-tech
 - Exclusively in C/C++ (mostly C-like C++)
- Worked on:
 - Lords of the Fallen
 - The Surge
 - Anno 2205
 - Portal Knights
 - Atlas Fallen
 - Unannounced Keen Games Title







High level vs. Low level optimizations

High level

- Reduction of work
- Algorithm improvements
- Finding better math formulas
- (Compiler optimizations)

High level vs. Low level optimizations

High level

- Reduction of work
- Algorithm improvements
- Finding better math formulas
- (Compiler optimizations)

Low level

- Better hardware utilization
- Cache aware programming
- "How can I use the hardware to more efficiently solve my problem?"

"I just want to program without having to worry about the hardware. If my program should run on different hardware I just recompile it and be done with it"

"I just want to program without having to worry about the hardware. If my program should run on different hardware I just recompile it and be done with it"

"The faster I ship software, the faster I make money - I don't care how long the program loads, the faster the program is finished, the better"

"I just want to program without having to worry about the hardware. If my program should run on different hardware I just recompile it and be done with it"

"The faster I ship software, the faster I make money - I don't care how long the program loads, the faster the program is finished, the better"

It might not feel like it, but today's software is *slow* if you take into account how insane the current hardware is. Try using an era appropriate Win98 machine and you'd be amazed

"I just want to program without having to worry about the hardware. If my program should run on different hardware I just recompile it and be done with it"

"The faster I ship software, the faster I make money - I don't care how long the program loads, the faster the program is finished, the better"

It might not feel like it, but today's software is *slow* if you take into account how insane the current hardware is. Try using an era appropriate Win98 machine and you'd be amazed

https://twitter.com/tsoding/status/1636036276687192068

This is where we're at right now (this is *not* a parody):



Let's work with a concrete example during this talk:

- Rotating a 2D image on the CPU with bilinear sampling
- Image is 4096x4096

Let's work with a concrete example during this talk:

- Rotating a 2D image on the CPU with bilinear sampling
- Image is 4096x4096

Bilinear Sampling:



Let's work with a concrete example during this talk:

- Rotating a 2D image on the CPU with bilinear sampling
- Image is 4096x4096

Input:

Bilinear Sampling:

- Unrotated 2D image
- Transformation matrix



Let's work with a concrete example during this talk:

- Rotating a 2D image on the CPU with bilinear sampling
- Image is 4096x4096

Input:

Bilinear Sampling:

- Unrotated 2D image
- Transformation matrix

Output:

• Rotated 2D image



Let's work with a concrete example during this talk:

- Rotating a 2D image on the CPU with bilinear sampling
- Image is 4096x4096

Input:

- Unrotated 2D image
- Transformation matrix

Output:

• Rotated 2D image

This example is run on a Intel i9-10980XE CPU

Bilinear Sampling:

Basically a 2D matrix transformation:

$$\begin{cases} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{cases}$$

- Look at every pixel in the output image
- Transform pixel coordinate using matrix to get coordinate in input image
- (bilinear sample pixel at coordinate)
- Write sampled pixel to output image





• The function is called in this context:

void main(const int argc, const char** argv) {

}

//assume input image is NxN (N==pow of 2)

const image* inputImage = loadImage(argv[1]); const float rotateAngleInRad = atof(argv[2]); float rotateTransform[4]; create2DRotateTransform(rotateTransform, rotateAngleInRad); image* outputImage = allocateEmptyImage(inputImage->size); timer rotateTimer = createTimer(); rotateTimer.start(); Rotate(outputImage, inputImage, rotateTransform); //<- this is our code rotateTimer.end(); printf("Rotation by %.lf degree took %.lfms\n", rotateAngleInRad, rotateTimer.timeInMilliSeconds()); return;

Naive version of rotate algorithm

```
void Rotate(image* outputImage, const image* inputImage, const float* rotateTransform) {
    const unsigned int size = inputImage.size;
    for( int y = 0; y < size; ++y ){
    for( int x = 0; x < size; ++x ){
        float xt = x, yt = y;
        Transform2D(&xt, &yt, rotateTransform);
        unsigned int sample = BilinearSampleAtPosition(xt, yt, inputImage);
        WriteSampleAtPosition(x, y, sample, outputImage);
    }
</pre>
```

Naive version of rotate algorithm

- It works
- Readable code
- ...

Performance baseline: Rotation by 22.5 degrees took 209.0ms (Should be run multiple times to get average)

• First optimization instinct?

Naive version of rotate algorithm

- It works
- Readable code
- ...

Performance baseline: Rotation by 22.5 degrees took 209.0ms (Should be run multiple times to get average)

- First optimization instinct?
 - Optimization efforts should *always* be based on data (Except for *super* obvious cases)

How do we know what we can improve on without knowing the hardware? -> Documentation (RTFM)

AMD: <u>https://gpuopen.com/ryzen-performance/</u>

Intel: <u>https://cdrdv2-public.intel.com/671488/248966-046A-software-optimization-manual.pdf</u>

ARM: <u>https://documentation-service.arm.com/static/5ed4bd67ca06a95ce53f917d?token=</u>

How do we know what we can improve on without knowing the hardware? -> Documentation (RTFM)

AMD: <u>https://gpuopen.com/ryzen-performance/</u>

Intel: <u>https://cdrdv2-public.intel.com/671488/248966-046A-software-optimization-manual.pdf</u>

ARM: <u>https://documentation-service.arm.com/static/5ed4bd67ca06a95ce53f917d?token=</u>

We'll focus on Intel for this talk.

It's in the CPU vendor's best interest not to have headlines like "My game runs slower on [CPU from vendor A] than on [CPU from vendor B] but the CPUs are the same speed!".

It's in the CPU vendor's best interest not to have headlines like "My game runs slower on [CPU from vendor A] than on [CPU from vendor B] but the CPUs are the same speed!".

You don't have to know the *whole* documentation but at the very least make yourself familiar with the lingo.

It's in the CPU vendor's best interest not to have headlines like "My game runs slower on [CPU from vendor A] than on [CPU from vendor B] but the CPUs are the same speed!".

You don't have to know the *whole* documentation but at the very least make yourself familiar with the lingo.

• Pulling out the SIMD hammer might not always be the first best solution.

It's in the CPU vendor's best interest not to have headlines like "My game runs slower on [CPU from vendor A] than on [CPU from vendor B] but the CPUs are the same speed!".

You don't have to know the *whole* documentation but at the very least make yourself familiar with the lingo.

- Pulling out the SIMD hammer might not always be the first best solution.
- Vendor specific tools will help you collect performance data
 - Intel V-Tune
 - \circ AMD uProf
 - Qualcomm Snapdragon Profiler

Running Intel V-Tune Microarchitecture Exploration to get broad idea of CPU utilization performance metrics of our program.

Wetterne > Rothine -									
Microarchitecture Exploration	Microarchitecture Exploration -	© th							
Anapos Cantgo anno Comedica (19) But	reary Botenup EretCo	Art Parters							
Grouping Function / Call Stock								:×[]%	1
Fahrbard / Call Black	CPUTime *	Clariffekk	Instructions Rolland	CPI Rate	Salary -	FortExtBand -	Bard Specialization (*	Sad-Feithand	Anne
b atta zili povarkas	409.000mg	1.568.000.000	2,439,000,000	0.007	- 10 4/1	21.22	83.75	0.05	
is atting other exception	100.000+++	803 000 000	1.117.000.000	0.909					
a stin loan decode block prist at	183.000mm	777.005.000	1,110,000,000	0.705	10.116	21.5%	9.42	44.24	
I HOW	INTERNAL COM	B72 (818 000)	1 005 (100-50)	1.000			-		
+ stow rtts hadd	-00.000mm B	291,000,000	\$19,000,000	0.581					
albie zito inter-	47.000ma 8	141.000.000	330.000.000	0.427	100.0%	75,2%	0.0%	0.01	
> sibi jpeg decode block prog du	32.000ms	135.000.000	80.000.000	1.505					
> athi idot ainid	30.000ms	126,000,000	258.000.000	0.405	d Die	0.075	0.0%	100 01	
> starts with	2% 000em #	102,000,000	138,000,000	0.738	87.1%	0.0%	47.1%	0.01	
a site crc12	20.000em 1	90.000.000	102,000,000	0.882		0.0%	0.0%	150.09	
> sibi parse entropy cooked data	19.000ms 1	80.000.000	80.000.000	1.000	103.2%	10.12%	1021.076	0.01	
> stbi jpeg huff decote	10.000mm 1	111,000,000	111,000,000	1.000	29.7%	28.5%	10 2%	3.05	
> memory	10.000mm 1	51,000.000	38.000.000	2.125	54.7%	6.17%	1710-026	35.25	
> RotateGrad	15.000min 1	48,000,000	75.000.000	0.640	68.8%	0.0%	0.2%	21.25	
s atta long get bit	15.000ms 1	80,000,000	#3.000.000	0.952	0.0%	0.005	0.0%	100.04	
> CasificatToUnt8	12.000mm 1	78,000.000	177.000.000	0.441	84.8%	-0.0%	0.0%	57.7%	
> stbi_gross_to/fer_unuefe	11.000mm 1	78,000.000	89,000,000	1.132	42.21	86.0%	42.5%	0.01	
+ tood jong image	11.000mm 1	42,000.000	96,000,000	0.438	0.0%	0.0%	10.0%	100.01	
> memory	0.000ms I	18,000,000	1.000.000	11200	4.0%	8.0%	10.0%	100.01	
> stbi jpag get bit	8.000mm	18,930,000	24,000.000	0.629	0.0%	10 AT	0.2%	100.01	
b alla_gall	6.000mm	15,971,000	33,000,000	0.438	0.0%	1.0%	100-3%	0.01	-
> sti jpeg detsardpe	7,000ems	45,000,000	24,000.000	1.873	0.0%	2.0%	0.0%	100.0%	
> sthi joeg fulf decode	4.200mm1	0,035,000	45,000.000	0.201	0.0%	0.0%	0.0%	100.01	
Http://wfragthapAllocFrienDontext	4.000zml	0,000,000	18,000,000	0.003	4.0%	8.0%	. 0.0%	100.04	
> MSDalateVite	3.000mm	3.000,000	24,000.000	6.125	0.091	0.0%	3.2%	100.04	
» MiConvertEntreLargePageToSmat	3-000mm	8,000,000	0.000.000	1,000	4.7%	0.075	0.0%	100.01	
RtEnterCriticalSection	2 (000mm)	0	0	0.000	4.0%	10.075	3.0%	100.01	
> etti joog finin	£ 000mm	3.011.000	18.000.000	0.967	4.0%	11.17%	0.2%	100.01	
NAAXWorkingGetEntree	2.000mm i	10011-000	27.000.000	0.221	0.0%		0.0%	100.0%	
PprtGnapPertomanceAccumulation	1 m000 1	10	-11	0.000	0.0%	12,016	D.0%	100.0%	
» memopy	1 000 mm	(1.000.000	-11		0.0%	3.0%	D.0%	100.01	
Bistabüsine Bistabüsine +	2.200em.)	0		0.001	0.0%	0.295	0.0%	100.04	
1.00 A.A.A.	and the second	1	-	2	1.00		1.525	100	-

Running Intel V-Tune Microarchitecture Exploration to get broad idea of CPU utilization performance metrics of our program.

Wetcome > Bittue -								
Microarchitecture Exploration 44	ananitrianian Exceptation -	0 11						
Antoni Castanama, Constantino, Anto	And Battern Freed Free	at Bigthese						
Counting Desition (Col Start)	all manut Training	an ranne.						Card of its
Grouping Purcease rule stack		I LONG I						
Fondton / Call Black	CPU Tires *	Couldtake	Instructions Rolland	OFI Rate	Steel	FortEclibour -	Bard Specialobon (*	Sad-EntBoard - Anti-
> 451_281_007presi	400.000ess #1	1,868,000.000	2,438,000,000	0.007	12.4%	21.2%	30.7%	0.0%
> stim_zib_courier	180.000ms 6.000	801,000,000	1,317.000.000	0.928				
> stil_jpeg_decode_block_prog_ac	183.000em (183.000em)	777,000,000	1.115.000.000	0.700	2012	21.7%	545	412%
	15T DODHU	B71,818,000	1,005,000,000					2
stam_100_0104	00.000###	291,000,000	\$19,000,000	0.581		1 ()		1. The second
> albietit_bhey	47.000ma	141.000.000	335,000,060	0.427	100.0%	- 75.2%	0.8%	0.0%
> sibi_jpeg_decode_block_prog_dt	32.000ms	135,000.000	80,000,000	1.505				
> stbiiddt_sired	30.000ms	120,000,000	258,000,000	0.405	4.0%	0.075	0.0%	100.015
> dbivzheth	2% 000ess 🖠	102,000,000	138,000,000	0.738	87.1%	0.0%	37.1%	-0 Pro
> size_crc32	20.000em	90.000.000	922.000.000	0.882	4.1%	10.1215	.0.1%	- 1503 (PR.
> stbi_parse_entropy_cooked_date	19.000ems	80,000.000	80.000.000	1.000	103.2%	1.2%	1021.0%	0.0%
> sbi_jpeg_huff_decote	18.000mm	111,000,000	111,000,000	1.000	29,7%	38.5%	101.2%	3.0%
> memory	10.000mm	51,000.000	38.000.000	2.125	54.7%	-8.195	110.0%	35.25
> RotateSmall	15.000min 1	48,000,000	75.000.000	0.645	-08.8%	0.0%	0.0%	21.2%
> atbi_ipeg_pet_bit	15.000ms 1	80,000.000	#3.000.000	0.952	0.0%	-0.005	0.0%	7(33.0%)
CasificalToUnt8	12.000ms	78,000.000	177.000.000	0.441	84.8%	-0.0%	0.0%	- 57.7%
#bi_grow_buffer_umafe	11.000mm 1	78,000,000	89,000,000	1.132	12.311	04.0%	42,5%	0.0%
> load_pog_image	11.000ma 1	42,000.000	96,000,000	0.438	0.011	0.0%	10.0%	100.0%
> memory	0.000ms I	18,000.000	1.000.000	11200	9,0%	0.014	10.0%	100-015
> athi_jpag_gat_bit	8.200mm	18,939,000	24,000.0mm	0.629	0.0%	3.0%	0.2%	1042.0%
h site_petb	6.200mm	15,971,000	33,000,000	0.493	0.0%	3.0%	100.0%	0.0%
> sti_jpeg_detsardpe	7,000mm	45.000.000	34,000.000	1.673	0.0%	=.0%	0.0%	100.0%
> etti_joeg_fuff_decode	4,000mm1	0,035,000	45,000.000	0.201	0.0%	0.0%	0.0%	100.0%
Http://www.interpage.com/pa	4.000zml	0,000,000	18,000,000	0.203	4.0%	8.0%	0.0%	100.0%
> MSDalateVia	3 000mm	3.090,000	24.000.000	61128	0.0%	0.0%	3.7%	100.0%
» MiConvertEntireLargePageToSmet	5-000mm	8,005,000	0.000.000	1,000	9.2%	6.7%	0.0%	100.0%
RtEnterCriticalSection	2 (000mm)	0	0	0.000	9.0%	10.075	3.0%	100,0%
> etti jong fivish	£-000mm	3.071.000	18.000.000	0.967	4.0%	11.0%	0.0%	100.0%
NAAXWorkingSetEntries	2:000mm	10.0111.000	27.000.000	0.221	0.0%	-11.0%	0.0%	100.0%
+ PprtGnapPerformanceAccumulation	[==000.2	(1)	-11	0.000	0.0%	10.016	D.3%	100.0%
> memopy	1 000mm	(8.000.000	- 11		0.0%	3.0%	D.0%	100.0%
> %avoigde1000ab88	2.200 mm.)	0	0	0.0010	0.0%	0.11%	0.0%	108.0%
with faith	and the second second	-	_	_				

Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation 测	Back-End Bound 🚿
Rotate	0.669	0.0%	0.0%	60.7%

Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation	Back-End Bound
Rotate	0.669	0.0%	0.0%	60.7%
	1			1

Strong indication of sub-optimal execution unit utilization

Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation 测	Back-End Bound
▶ Rotate	0.669	0.0%	0.0%	60.7%
	↑			†

Strong indication of sub-optimal execution unit utilization

CPI: Clocks per Instruction, the lower the better.



Strong indication of sub-optimal execution unit utilization

CPI: Clocks per Instruction, the lower the better.

High-Level Explanation of Front- and Back-End:



Strong indication of sub-optimal execution unit utilization

CPI: Clocks per Instruction, the lower the better.

High-Level Explanation of Front- and Back-End:

• Front-End: Transforms ASM into u-Ops



Strong indication of sub-optimal execution unit utilization

CPI: Clocks per Instruction, the lower the better.

High-Level Explanation of Front- and Back-End:

- Front-End: Transforms ASM into u-Ops
- Back-End: Issues u-Ops

What are Execution Units?

What are Execution Units?

• Most (all?) modern CPUs are superscalar CPUs.
What are Execution Units?

- Most (all?) modern CPUs are superscalar CPUs.
- That means that they can execute a certain set of instructions in parallel (instruction-level parallelism).

What are Execution Units?

- Most (all?) modern CPUs are superscalar CPUs.
- That means that they can execute a certain set of instructions in parallel (instruction-level parallelism).
- What instructions can be executed in parallel is determined by what execution units are available.

What are Execution Units?

- Most (all?) modern CPUs are superscalar CPUs.
- That means that they can execute a certain set of instructions in parallel (instruction-level parallelism).
- What instructions can be executed in parallel is determined by what execution units are available.
- Prerequisite: No dependencies

Intel[®] 64 and IA-32 Architectures Optimization Reference Manual Chapter 2.3.1.2

Port 0	Port 1 ¹	Port 2	Port 3	Port 4	Port 5 ²	Port 6	Ports 7, 8	Port 9	Port 10	Port 11
INT ALU LEA INT Shift Jump1	INT ALU LEA INT Mul INT Div	Load	Load	Store Data	INT ALU LEA INT MUL Hi	INT ALU LEA INT Shift Jump2	Store Address	Store Data	int alu Lea	Load
FMA Vec ALU Vec Shift FP Div	FMA* Fast Adder* Vec ALU* Vec Shift* Shuffle*				FMA** Fast Adder Vec ALU Shuffle					

Table 2-1. Dispatch Port and Execution Stacks of the Golden Cove Microarchitecture

NOTES:

1. "*" in this table indicates that these features are not available for 512-bit vectors.

2. **** in this table indicates that these features are not available for 512-bit vectors in Client parts.

Intel[®] 64 and IA-32 Architectures Optimization Reference Manual Chapter 2.3.1.2

Port 0	Port 1 ¹	Port 2	Port 3	Port 4	Port 5 ²	Port 6	Ports 7, 8	Port 9	Port 10	Port 11
INT ALU LEA INT Shift Jump1	INT ALU LEA INT Mul INT Div	Load	Load	Store Data	INT ALU LEA INT MUL Hi	INT ALU LEA INT Shift Jump2	Store Address	Store Data	int alu Lea	Load
FMA Vec ALU Vec Shift FP Div	FMA* Fast Adder* Vec ALU* Vec Shift* Shuffle*				FMA** Fast Adder Vec ALU Shuffle					

Table 2-1. Dispatch Port and Execution Stacks of the Golden Cove Microarchitecture

NOTES:

1. "*" in this table indicates that these features are not available for 512-bit vectors.

2. **** in this table indicates that these features are not available for 512-bit vectors in Client parts.

Gives you an idea of what instructions can be parallelized

Going back to our naive example:

```
void Rotate(image* outputImage, const image* inputImage, const float* rotateTransform) {
 const unsigned int size = inputImage.size;
 for(int y = 0; y < size; ++y){
 for( int x = 0; x < size; ++x ){
   float xt = x, yt = y;
   Transform2D(&xt, &yt, rotateTransform);
   unsigned int sample = BilinearSampleAtPosition(xt, yt, inputImage);
   WriteSampleAtPosition(x, y, sample, outputImage);
```

Going back to our naive example:

void Rotate(image* outputImage, const image* inputImage, const float* rotateTransform) {
 const unsigned int size = inputImage.size;
 for(int y = 0; y < size; ++y){
 for(int x = 0; x < size; ++x){
 float xt = x, yt = y;
 Transform2D(&xt, &yt, rotateTransform);
 unsigned int sample = BilinearSampleAtPosition(xt, yt, inputImage);
 WriteSampleAtPosition(x, y, sample, outputImage);
 }
</pre>

Loop unrolling to the rescue!

Loop unrolling to the rescue!

• Each loop iteration is independent of each other so this works out nicely

Loop unrolling to the rescue!

- Each loop iteration is independent of each other so this works out nicely
- Naive implementation with 4x loop unrolling:

Loop unrolling to the rescue!

- Each loop iteration is independent of each other so this works out nicely
- Naive implementation with 4x loop unrolling:
- Also added specialized functions that work on 4 elements instead of 1

Loop unrolling to the rescue!

- Each loop iteration is independent of each other so this works out nicely
- Naive implementation with 4x loop unrolling:
- Also added specialized functions that work on 4 elements instead of 1

```
void Rotate(image* outputImage, const image* inputImage, const float* rotateTransform) {
    const unsigned int size = inputImage.size;
```

```
for( int y = 0; y < size; ++y ){
```

```
for( int x = 0; x < size; x += 4 ){
```

```
float[] xt = {x+0,x+1,x+2,x+3}, yt = {y, y, y, y};
```

unsigned int samples[4];

Transform2DMultiple4(&xt, &yt, rotateTransform); BilinearSamplesAtPositions4(xt, yt, samples, inputImage);

WriteSamplesAtPositions4(xt, yt, samples, outputImage);

What does V-Tune say?

Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation	Back-End Bound
▶ Rotate	0.669	0.0%	0.0%	60.7%
	VS			
Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation »	Back-End Bound »
Rotate	0.358	1.0%	0.0%	5.9%

What does V-Tune say?

Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation	Back-End Bound
Rotate	0.669	0.0%	0.0%	60.7%
	VS			
Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation »	Back-End Bound
Rotate	0.358	1.0%	0.0%	5.9%

• Better execution unit utilization

What does V-Tune say?

Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation	Back-End Bound
Rotate	0.669	0.0%	0.0%	60.7%
, ,	VS			
Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation »	Back-End Bound
Rotate	0.358	1.0%	0.0%	5.9%

- Better execution unit utilization
- Better code generation by the compiler

What does V-Tune say?

Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation	Back-End Bound
Rotate	0.669	0.0%	0.0%	60.7%
, ,	VS			
Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation »	Back-End Bound
Rotate	0.358	1.0%	0.0%	5.9%

- Better execution unit utilization
- Better code generation by the compiler
- Still readable

Rotation by 22.5 degrees took 209.0ms

What does V-Tune say?

Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation	Back-End Bound
Rotate	0.669	0.0%	0.0%	60.7%
, ,	VS			
Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation »	Back-End Bound
Rotate	0.358	1.0%	0.0%	5.9%

- Better execution unit utilization
- Better code generation by the compiler
- Still readable
- Only minimal changes needed

Rotation by 22.5 degrees took 151.9ms

Lets look at memory accesses and cache utilization using V-Tune Memory Access Analysis

Function / Call Stack	Memory Bound >>>	LLC Miss Count
▶ Rotate	14.2%	1,400,098

Ouch

Lets look at memory accesses and cache utilization using V-Tune Memory Access Analysis

Function / Call Stack	Memory Bound »	LLC Miss Count
▶ Rotate	14.2%	1,400,098

Ouch

What could be the reason?

Excourse CPU Caches (High level overview):

Excourse CPU Caches (High level overview):

CPU Cache

KBytes/MBytes

Excourse CPU Caches (High level overview):

CPU Cache

Main Memory

KBytes/MBytes

______GBytes

Excourse CPU Caches (High level overview):

CPU Cache Main Memory KBytes/MBytes GBytes This image has been loaded into memory:



Excourse CPU Caches (High level overview):

CPU Cache Main Memory KBytes/MBytes GBytes This image has been loaded into memory:

Excourse CPU Caches (High level overview):





Excourse CPU Caches (High level overview):



Main Memory Assume we want to access one pixel after another

Excourse CPU Caches (High level overview):



Main MemoryAssume we want to access one pixel after anotherFor each access, the CPU first checks the cache

Excourse CPU Caches (High level overview):



Main Memory Assume we want to access one pixel after another For each access, the CPU first checks the cache If the data is not in the cache, it gets accessed from main memory. But instead of just accessing the one pixel, it moves a cache-line into the cache.

This image has been loaded into memory:



Excourse CPU Caches (High level overview):



Main Memory Assume we want to access one pixel after another For each access, the CPU first checks the cache If the data is not in the cache, it gets accessed from main memory. But instead of just accessing the one pixel, it moves a cache-line into the cache.

This image has been loaded into memory:



Excourse CPU Caches (High level overview):



Main Memory Assume we want to access one pixel after another For each access, the CPU first checks the cache If the data is not in the cache, it gets accessed from main memory. But instead of just accessing the one pixel, it moves a cache-line into the cache.

This is known as a cache miss

This image has been loaded into memory:



According to Intel[®] 64 and IA-32 Architectures Software Developer's Manual Chapter 12.1, a cache line is 64 bytes

According to Intel[®] 64 and IA-32 Architectures Software Developer's Manual Chapter 12.1, a cache line is 64 bytes

This is done because it is assumed that you're also interested in neighboring data and not just one byte (or pixel in this case)

According to Intel[®] 64 and IA-32 Architectures Software Developer's Manual Chapter 12.1, a cache line is 64 bytes

This is done because it is assumed that you're also interested in neighboring data and not just one byte (or pixel in this case)

Prefetcher within the CPU will fetch next cache lines in advance if a sequential access pattern is detected.

CPU has multiple caches in a hierarchy:

- L1 cache per core (very small and very fast)
- L2 cache shared between cores (larger and slower)
- L3 cache shared between cores (largest and slowest)

	CPU Core Registers L1 Cache (on chip, banked)	
`	L2 Cache Unified	
,	L3 Cache (Unified)	
	Main Memory	

https://en.wikipedia.org/wiki/Cache_hierarchy

Cache MissCache Hit



CPU Cache



KBytes/MBytes GBytes
GBytes
Cache Miss
Cache Hit
CPU Cache



KBytes/MBytes

GBytes Cache Miss Cache Hit

Assume code like this:



Assume code like this:

CPU Cache Main Memory

KBytes/MBytes GBytes Cache Miss Cache Hit

Assume code like this:

CPU Cache

Main Memory

KBytes/MBytes

GBytes Cache Miss Cache Hit

Assume code like this:

CPU Cache



KBytes/MBytes

GBytes Cache Miss Cache Hit

Assume code like this:

CPU Cache



KBytes/MBytes

GBytes Cache Miss Cache Hit

Assume code like this:

CPU Cache



KBytes/MBytes

GBytes Cache Miss Cache Hit

Assume code like this:



Assume code like this:



Assume code like this:

for(int i= 0; i < image.size*image.size; ++i) {
 //Do something with this pixel...
 DoSomething(image.pixel[i]);</pre>



Assume code like this:

for(int i= 0; i < image.size*image.size; ++i) {
 //Do something with this pixel...
 DoSomething(image.pixel[i]);</pre>



Assume code like this:

for(int i= 0; i < image.size*image.size; ++i) {
 //Do something with this pixel...
 DoSomething(image.pixel[i]);</pre>



Assume code like this:

for(int i= 0; i < image.size*image.size; ++i) {
 //Do something with this pixel...
 DoSomething(image.pixel[i]);</pre>



Assume code like this:

for(int i= 0; i < image.size*image.size; ++i) {
 //Do something with this pixel...
 DoSomething(image.pixel[i]);</pre>



Prefetcher fetches next cache line because of the sequential access pattern

Assume code like this:

Let's revisit the algorithm:

Let's revisit the algorithm: We want to rotate this image by 50°:



Let's revisit the algorithm: We want to rotate this image by 50°:





Let's revisit the algorithm: We want to rotate this image by 50°:





• Write access is sequential, no problem here

Let's revisit the algorithm: We want to rotate this image by 50°:



- Write access is sequential, no problem here
- Read access is non-sequential
 - \circ Worst case: every read is a cache miss

Let's revisit the algorithm: We want to rotate this image by 50°:



- Write access is sequential, no problem here
- Read access is non-sequential
 - \circ $\;$ Worst case: every read is a cache miss $\;$

What can we do about it?

Answer: apply loop blocking (aka strip-mining for 1D data sets) to make access pattern more local

Answer: apply loop blocking (aka strip-mining for 1D data sets) to make access pattern more local

Intel® 64 and IA-32 Architectures Optimization Reference Manual Chapter 5.5.3



```
constexpr int blockSize = 64;
void RotateImageBlock(const int startX, const int startY, image* outputImage, const image* inputImage, const float* rotateTransform) {
 for( int y = startY; y < startY + blockSize; ++y ) {</pre>
 for( int x = startX; x < startX + blockSize; x + = 4)
   float xt[] = {x+0,x+1,x+2,x+3}, yt[] = {y, y, y, y};
   unsigned int samples[4];
   Transform2DMultiple4(&xt, &yt, rotateTransform);
   BilinearSamplesAtPositions4(xt, yt, samples, inputImage);
   WriteSamplesAtPositions4(xt, yt, samples, outputImage);
void Rotate(image* outputImage, const image* inputImage, const float* rotateTransform, int size){
 for(int y = 0; y < size; y += blockSize) {
 for(int x = 0; x < size; x += blockSize) {
   RotateImageBlock(x, y, outputImage, inputImage, rotateTransform);
```

```
constexpr int blockSize = 64;
void RotateImageBlock(const int startX, const int startY, image* outputImage, const image* inputImage, const float* rotateTransform) {
 for( int y = startY; y < startY + blockSize; ++y ) {</pre>
 for( int x = startX; x < startX + blockSize; x + = 4)
   float xt[] = {x+0,x+1,x+2,x+3}, yt[] = {y, y, y, y};
   unsigned int samples[4];
   Transform2DMultiple4(&xt, &yt, rotateTransform);
   BilinearSamplesAtPositions4(xt, yt, samples, inputImage);
   WriteSamplesAtPositions4(xt, yt, samples, outputImage);
void Rotate(image* outputImage, const image* inputImage, const float* rotateTransform, int size){
 for(int y = 0; y < size; y += blockSize) {
```

```
for (int x = 0; x < size; x += blockSize) {
```

RotateImageBlock(x, y, outputImage, inputImage, rotateTransform);

```
constexpr int blockSize = 64;
void RotateImageBlock(const int startX, const int startY, image* outputImage, const image* inputImage, const float* rotateTransform) {
 for( int y = startY; y < startY + blockSize; ++y ) {</pre>
 for( int x = startX; x < startX + blockSize; x + = 4)
   float xt[] = {x+0,x+1,x+2,x+3}, yt[] = {y, y, y, y};
   unsigned int samples[4];
   Transform2DMultiple4(&xt, &yt, rotateTransform);
   BilinearSamplesAtPositions4(xt, yt, samples, inputImage);
   WriteSamplesAtPositions4(xt, yt, samples, outputImage);
void Rotate(image* outputImage, const image* inputImage, const float* rotateTransform, int size){
```

```
for(int y = 0; y < size; y += blockSize) {
  for(int x = 0; x < size; x += blockSize) {
</pre>
```

RotateImageBlock(x, y, outputImage, inputImage, rotateTransform);

What does V-Tune say?

Function / Call Stack	Memory Bound »	LLC Miss Count
▶ Rotate	14.2%	1,400,098
VS		
Function / Call Stack	Memory Bound »	LLC Miss Count
Rotate	2.7%	0

• Again, only minimal code changes needed

Function / Call Stack	Memory Bound »	LLC Miss Count
▶ Rotate	14.2%	1,400,098
VS		
Function / Call Stack	Memory Bound »	LLC Miss Count
▶ Rotate	2.7%	0

- Again, only minimal code changes needed
- Experiment with block size but 64 should be a good first guess

Function / Call Stack	Memory Bound »	LLC Miss Count
▶ Rotate	14.2%	1,400,098
VS		
Function / Call Stack	Memory Bound »	LLC Miss Count
▶ Rotate	2.7%	0

- Again, only minimal code changes needed
- Experiment with block size but 64 should be a good first guess
- Have to be careful to handle cases where image size < block size

Function / Call Stack	Memory Bound »	LLC Miss Count
▶ Rotate	14.2%	1,400,098
VS		
Function / Call Stack	Memory Bound »	LLC Miss Count
▶ Rotate	2.7%	0

- Again, only minimal code changes needed
- Experiment with block size but 64 should be a good first guess
- Have to be careful to handle cases where image size < block size
- Perfect setup for next optimization

What does V-Tune say?

Function / Call Stack	Memory Bound »	LLC Miss Count
▶ Rotate	14.2%	1,400,098
VS		
Function / Call Stack	Memory Bound	LLC Miss Count
▶ Rotate	2.7%	0

- Again, only minimal code changes needed
- Experiment with block size but 64 should be a good first guess
- Have to be careful to handle cases where image size < block size
- Perfect setup for next optimization

Rotation by 22.5 degrees took 151.9ms

Function / Call Stack	Memory Bound »	LLC Miss Count
▶ Rotate	14.2%	1,400,098
VS		
Function / Call Stack	Memory Bound »	LLC Miss Count
Rotate	2.7%	0

- Again, only minimal code changes needed
- Experiment with block size but 64 should be a good first guess
- Have to be careful to handle cases where image size < block size
- Perfect setup for next optimization

Rotation by 22.5 degrees took 123.9ms

• So far we only used one core

- So far we only used one core
- All modern CPUs have multiple cores

- So far we only used one core
- All modern CPUs have multiple cores
- Nowadays you *have* to know how to utilize multiple cores *if* your domain is performance sensitive

- So far we only used one core
- All modern CPUs have multiple cores
- Nowadays you *have* to know how to utilize multiple cores *if* your domain is performance sensitive
- Lots of traps to fall into

- So far we only used one core
- All modern CPUs have multiple cores
- Nowadays you *have* to know how to utilize multiple cores *if* your domain is performance sensitive
- Lots of traps to fall into
- Rule of thumb for multithreading code that shares data:
 - Better to have something that works than something that's fast (finding and fixing multithreading bugs require good debug skills)
- Job system lends itself perfectly for this use case
 - General idea: break work down into independent jobs, assign threads as workers, each worker works on one job

- Job system lends itself perfectly for this use case
 - General idea: break work down into independent jobs, assign threads as workers, each worker works on one job
- One producer, multiple consumer
 - Main thread creates work, worker consume work

- Job system lends itself perfectly for this use case
 - General idea: break work down into independent jobs, assign threads as workers, each worker works on one job
- One producer, multiple consumer
 - Main thread creates work, worker consume work



- Job system lends itself perfectly for this use case
 - General idea: break work down into independent jobs, assign threads as workers, each worker works on one job
- One producer, multiple consumer
 - Main thread creates work, worker consume work
- What would be a good granularity for a job?
 - Loop box optimization makes this obvious



Quick overview of things we have to do to add a job system

- Create worker threads
- Create independent jobs
- Schedule jobs
- Wait until all jobs are finished

Create worker threads

- Ideally utilize all cores find out how many cores exist
 - Use std::thread::hardware_concurrency() if you use C++11 or newer
 - Use OS specific functions if you use C or an earlier C++ standard
 - \circ GetLogicalProcessorInformation() for win32
 - get_nprocs() for posix

```
std::thread** CreateWorker(SharedWorkerData* workerData){
    unsigned int workerCount = std::thread::hardware_concurrency()-lu;
    std::thread** worker = new std::thread*[workerCount];
    for(int i = 0; i < workerCount; ++i){
        worker[i] = new std::thread(&WorkerMain, workerData);
    }
    return worker;
}</pre>
```

Create independent jobs

• Group job data into new data structure

};

struct RotateJobData {
 image* outputImage;
 const image* inputImage;
 const float* rotateTransform;
 int startX;
 int startY;

• Create shared data for all worker

struct SharedWorkerData {
 int jobCount;
 RotateJobData* jobs;
 std::mutex* jobLock;

];

SharedWorkerData* CreateSharedWorkerData(int blockSize, const image* inputImage, image* outputImage, const float* rotateTransform) { const int jobCount = inputImage->size / blockSize; SharedWorkerData* sharedWorkerData = new SharedWorkerData; sharedWorkerData->jobCount = imageSize / blockSize; sharedWorkerData->jobLock = new std::mutex(); sharedWorkerData->jobs = new RotateJobData[jobCount]; for(int i = 0; i < jobCount; ++i){</pre>

sharedWorkerData->jobs[i].outputImage = outputImage; sharedWorkerData->jobs[i].inputImage = inputImage; sharedWorkerData->rotateTransform = rotateTransform; sharedWorkerData->startX = x; sharedWorkerData->startY = y; x += blockSize;

if(x > size) { x = **0**; y += blockSize; }

return sharedWorkerData;

• Finally, add worker function that does the work

```
void WorkerMain(SharedWorkerData* sharedData){
  while(true){
    RotateJobData* jobData;
    if(sharedData->jobLock.lock()) {
        if(sharedData->jobCount == 0)
            return;
        jobData = &sharedData[sharedData->jobCount--];
        sharedData->jobLock.unlock();
    }
    RotateImageBlock(jobData->startX, jobData->startY, jobData->outputImage,
        jobData->inputImage, jobData->rotateTransform);
```

- Rotate function now just has to schedule the jobs
 - Also helps with work
 - After that, waits for all workers to finish

void Rotate(image* outputImage, const image* inputImage, const float* rotateTransform){
 const int blockSize = 64;

SharedWorkerData* sharedWorkerData = CreateSharedWorkerData(blockSize, inputimage,

outputImage, rotateTransform);

```
std::thread** workers = CreateWorker(sharedWorkerData);
```

```
WorkerMain(sharedWorkerData);
```

```
for(int i = 0; i < std::thread::hardware_concurrency-lu; ++i){
    workers[i]->join();
```

- Rotate function now just has to schedule the jobs
 - Also helps with work
 - After that, waits for all workers to finish

void Rotate(image* outputImage, const image* inputImage, const float* rotateTransform){
 const int blockSize = 64;

SharedWorkerData* sharedWorkerData = CreateSharedWorkerData(blockSize, inputimage,

outputImage, rotateTransform);

```
std::thread** workers = CreateWorker(sharedWorkerData);
```

```
WorkerMain(sharedWorkerData);
```

```
for(int i = 0; i < std::thread::hardware_concurrency-lu; ++i){
    workers[i]->join();
```

Rotation by 22.5 degrees took 123.9ms

- Rotate function now just has to schedule the jobs
 - Also helps with work
 - After that, waits for all workers to finish

void Rotate(image* outputImage, const image* inputImage, const float* rotateTransform){
 const int blockSize = 64;

SharedWorkerData* sharedWorkerData = CreateSharedWorkerData(blockSize, inputimage,

outputImage, rotateTransform);

```
std::thread** workers = CreateWorker(sharedWorkerData);
```

```
WorkerMain(sharedWorkerData);
```

```
for(int i = 0; i < std::thread::hardware_concurrency-lu; ++i){
    workers[i]->join();
```

Rotation by 22.5 degrees took 10.7ms ms

- If you're using an existing engine or framework, job system is most likely already in place
 - Eg: Job System in Unity <u>https://docs.unity3d.com/Manual/JobSystem.html</u>
- Multiple job systems with different granularities not uncommon
 - Jobs that have to finish this frame (will block if not finished by end of frame)
 - Jobs that can run over multiple frames without blocking

• Many traps to fall into

- Many traps to fall into
 - False sharing
 - Race conditions
 - Deadlocks

(Performance) (Behavior) (Crashes)

- Many traps to fall into
 - False sharing (Performance)
 - Race conditions (Behavior)
 - Deadlocks (Crashes)
- Make data sharing between threads as simple as possible
 - Simple queue will fit most use cases

- Many traps to fall into
 - False sharing (Performance)
 - Race conditions (Behavior)
 - Deadlocks (Crashes)
- Make data sharing between threads as simple as possible
 - Simple queue will fit most use cases
- Requirements might change between platforms
 - Eg: Busy-waiting on PC more acceptable than on mobile (battery life)

SIMD = \underline{S} ingle \underline{I} nstruction \underline{M} ultiple \underline{D} ata

SIMD = \underline{S} ingle \underline{I} nstruction \underline{M} ultiple \underline{D} ata

Instruction Set + Registers that work on multiple pieces of data at once

SIMD = \underline{S} ingle \underline{I} nstruction \underline{M} ultiple \underline{D} ata

Instruction Set + Registers that work on multiple pieces of data at once

Example multiplying numbers:

SIMD = \underline{S} ingle \underline{I} nstruction \underline{M} ultiple \underline{D} ata

Instruction Set + Registers that work on multiple pieces of data at once

Example multiplying numbers:

Scalar:

```
void scalarMul(float* values, float multiplier)
{
    values[0] *= multiplier;
    values[1] *= multiplier;
    values[2] *= multiplier;
    values[3] *= multiplier;
```

SIMD = \underline{S} ingle \underline{I} nstruction \underline{M} ultiple \underline{D} ata

Instruction Set + Registers that work on multiple pieces of data at once

Example multiplying numbers:

Scalar:

```
void scalarMul(float* values, float multiplier)
{
    values[0] *= multiplier;
    values[1] *= multiplier;
    values[2] *= multiplier;
    values[3] *= multiplier;
```

SIMD:

void simdMul(float* values, float multiplier)

__**m128** val = _mm_load_ps(values); __**m128** mul = _mm_set_ps1(multiplier);

__**m128** res = _mm_mul_ps(val, mul); _mm_store_ps(values, res);

Generally also called "Vectorization"

Compilers have a feature called "Auto-Vectorization" that *theoretically* detects code that can be transformed to be used with SIMD intrinsic.

Can we rely on the compiler's auto-vectorization?

Can we rely on the compiler's auto-vectorization?

"The compiler will optimize it!"

Can we rely on the compiler's auto-vectorization?

"The compiler will optimize it!"



• Gut feeling: 4x loop unrolled version should be trivial to auto-vectorize

- Gut feeling: 4x loop unrolled version should be trivial to auto-vectorize
 - Gut feeling doesn't count, let's check the data

- Gut feeling: 4x loop unrolled version should be trivial to auto-vectorize
 - Gut feeling doesn't count, let's check the data
 - Either look at the generated ASM code or check V-Tune

- Gut feeling: 4x loop unrolled version should be trivial to auto-vectorize
 - Gut feeling doesn't count, let's check the data
 - Either look at the generated ASM code or check V-Tune
- V-Tune HPC Performance Characterization tells us the harsh truth:

• Gut feeling: 4x loop unrolled version should be trivial to auto-vectorize

- Gut feeling doesn't count, let's check the data
- Either look at the generated ASM code or check V-Tune

• V-Tune HPC Performance Characterization tells us the harsh truth:

Function / Call Stack	CPU Time 🔻 🔍			Momory Bound	Voctorization
	Effective Time	Spin Time	Overhead Time	Welliory Bound	Vectorization
Rotate	0.232s	0s	0s	3.7%	0.0%

Compiled with msvc 19.33.31630 (ships with VS2022) with compiler options -**O2** -arch:avx2

Mileage may vary with a different compiler

• Gut feeling: 4x loop unrolled version should be trivial to auto-vectorize

- Gut feeling doesn't count, let's check the data
- Either look at the generated ASM code or check V-Tune

• V-Tune HPC Performance Characterization tells us the harsh truth:

Function / Call Stack	CPU Time 🔻 🔍			Momony Round	Voctorization
	Effective Time	Spin Time	Overhead Time	Wentory Bound	Vectorization
Rotate	0.232s	0s	0s	3.7%	0.0%

Compiled with msvc 19.33.31630 (ships with VS2022) with compiler options -**O2** -arch:avx2

Mileage may vary with a different compiler

Let's check the ASM for good measure (compiled with msvc flags -**O2** -arch:AVX2)

Let's check the ASM for good measure (compiled with msvc flags -**O2** -arch:AVX2)

```
void Transform2DMultiple4(float* x, float* y,
            const float* mat)
{
  for(int i = 0; i < 4; ++i)
     float xx = x[i] * mat[0] + y[i] * mat[1];
     float yy = x[i] * mat[2] + y[i] * mat[3];
     x[i] = xx;
     y[i] = yy;
```

Let's check the ASM for good measure (compiled with msvc flags -**O2** -arch:AVX2)

```
void Transform2DMultiple4(float* x, float* y,
            const float* mat)
  for(int i = 0; i < 4; ++i)
     float xx = x[i] * mat[0] + y[i] * mat[1];
     float yy = x[i] * mat[2] + y[i] * mat[3];
     x[i] = xx;
     y[i] = yy;
```

vold Transform	2DMultiple4(float ",float ",float const ") PF
*OV	ecs, DACRD FTH _mat\$[esp-4]
mov	edx, CMCRD FTR _x\$[esp+4]
ROY	esx, DADRD PTR _st[esp-4]
MOVEE	weed, GAGND FTR [ecs+12]
BOVSE	NEWS, CHORD FTR [odx]
ROVES	xeel, DADHD PTR [max]
POVEDS	anno, anno
Mulse	xmm3, CNORD PTR [ecx]
mulas	unnel, CACHD PTR [eco+0]
mulss	intel, stati
sules	weel, DADED FTR [acs+4]
addee	amm2, arms0
addss	ame3, and
NOVES	DWOHD FIR [edx], xmm3
ROVER	icen1, OMORD PTR [eax+4]
ROVES	DWORD FTR [eax], are2
TOVER	xmm2, DMCRD PTR [wcs+12]
POVSE	xmm1, CACHD PTR [adx+4]
sulss	anel, spel
ROVADE	Emmy Barry
mulss	xmm3, CNORD FTR [ecs]
sulss	and, OHOND PTR [ecs+4]
sulse	wmet, ChOHD PTR [scs+0]
addss	sma, smil
sidin	10002, 10000
POVER	DWDRD FTR [edu#4], smm2
REVER	Annal, CANCED FTR [max+8]
TOVER	DWDRD FTR [eas+4], ann2
MOVSS	xmmB, DWORD PTR [adx+6]
ROVES	xmm2, CMCHD PTR [ecx+12]
POVEDS	send, and
sulss	New3, CHOND PTR [ecs]
culas	small, ChOND PTR [scs+6]
rulas	ann2, sumi-
wulse	xmm1, DAGRD FTR [ecx+4]
addas	ann2, ann8
addss	send, send
ROVAL	DWGRD PTR [edx+S], smm3
BOVEE	see1, DACHD PTR [max+12]
HOVES	DWORD FTR [eax+8], smm2
BOVER	weed, CMOND PTR [eds+12]
POVSE	Amm2, CACHD FTR [acvel2]
ROVADE	same, samt
sular	anni, CACHD FTR [ecx]
mulss	xered, DACHD FTR [ecx+8]
mulss	sam2, smel
mulas	ares1, CADHD FTR [acx+4]
addss	yeard, aread
addas	anni, anni
POVER	DWDHD FTR [adx+12], semi
BOVES	CADRO FTR [cox+12], smm2
ret.	
Let's check the ASM for good measure (compiled with msvc flags -O2 -arch:AVX2)

```
void Transform2DMultiple4(float* x, float* y,
          const float* mat)
for(int i = 0; i < 4; ++i)
   float xx = x[i] * mat[0] + y[i] * mat[1];
   float yy = x[i] * mat[2] + y[i] * mat[3];
   x[i] = xx;
   y[i] = yy;
```

old Tr	ans form2	DMultiple4(float ",float ",float const ") PROC	
	*0V	ecs, DACHD FTH _matf[exp-4]	
	MOV.	edx, DWDRD FTR _x\$[exp+4]	
	ROY	eax, GADRD PTR[eap-4]	
	#OVSE	weed, choud PTR [ecs+12]	
	ROVSE	NEWS, CHICHE FTR [odx]	
	ROVES	weel, DADHD FTR [max]	
	movaps.	xmm0, xxm3	
	mulse	xmm3, CNCRD PTR [ecx]	
	mulas	emmil, CACHED FIR [scs+0]	
	mulss	keel, keel	
	sulss	ame1, DAGNE PTR [ecs+4]	
	addee	xmm2, xmm0	
	addss	xme3, xmm1	
	ROVES	DWOHD FIR [edx], xmm3	
	ROVER	kmm1, DWOHD PTR [eax+4]	
	ROVIS	DWORD FTR [dax], smm2	
	ROVER	xmm2, DWGRD PTR [mcs+12]	
	ROVER	NUMEZ, CACHO PTR [adx+4]	
	sulss	smal, smal	
	SOVADE	smil, smil	
	Hulss	NHMB3, CWOHD PTR [ecs]	
	sulss	wammi, OWORD FIR [ecs+4]	
	mulse	xmm#, CACHD FTR [ecx+8]	
	addss	see, see	
	addax	NEWS, NEWS	•
	ROVER	DADED ALK [edno-1" and	А
	WILVER.	sent, could sented	
	POVER	towned with [east-s]; since	
	ROVSS	when the second with formation	
	NOV55	Name and All [scatter]	
	mules	and Add the Local	
	eulee	week, choice FTR [access]	
	-ulue	name a second	
	sules	west, DAGED PTR [ecs+4]	
	addas	ann2, ann0	
	addss	send, senti	
	BOVER	DWORD PTR [edx+8], and	
	NOVEL	ammi, DACHD PTR [max+12]	
	NOVES	DWORD FTR [eax+8], smm2	
	BOVES	west, DACHD PTR [eds+12]	
	20155	Amm2, CAGHD FTR [acve12]	
	ROVADE	smm0, smm3	
	sulse	anni, DACHD FTR [ecx]	
	mulss	xee8, DACRD PTR [ecx+8]	
	mulss	sem2, semi	
	mulas	and, CMCHO PTR [acx+4]	
	addss	see2, seed	
	addes	anni, anni	
	POVER	DWDHD FTR [edx+12], see1	
	MOVES	CHORD FTR [cos+12], snm2	
	ret.		

All scalar :(

void Transform2DMultiple4(float* x, float* y, const float* mat)

__**m128** xx = _mm_load_ps(x); __**m128** yy = _mm_load_ps(y);

__m128 mat00 = _mm_set_ps1(mat[0]); __m128 mat01 = _mm_set_ps1(mat[1]); __m128 mat10 = _mm_set_ps1(mat[2]); __m128 mat11 = _mm_set_ps1(mat[3]);

__**m128** xxx = _mm_add_ps(_mm_mul_ps(xx, mat00), _mm_mul_ps(yy, mat01));

__**m128** yyy = _mm_add_ps(_mm_mul_ps(xx, mat10), _mm_mul_ps(yy, mat11));

_mm_store_ps(x, xxx); _mm_store_ps(y, yyy);

(compiled with msvc flags -O2)

void Transform2DMultiple4(float* x, float* y, const float* mat)

__**m128** xx = _mm_load_ps(x); __**m128** yy = _mm_load_ps(y);

__m128 mat00 = _mm_set_ps1(mat[0]); __m128 mat01 = _mm_set_ps1(mat[1]); __m128 mat10 = _mm_set_ps1(mat[2]); __m128 mat11 = _mm_set_ps1(mat[3]);

__**m128** xxx = _mm_add_ps(_mm_mul_ps(xx, mat00), _mm_mul_ps(yy, mat01));

__**m128** yyy = _mm_add_ps(_mm_mul_ps(xx, mat10), _mm_mul_ps(yy, mat11));

_mm_store_ps(x, xxx); _mm_store_ps(y, yyy);

(compiled with msvc flags -O2)

void Transform2	2DMultiple4(float *,float *,float const *) PROC
mov	eax, DWORD PTR _mat\$[esp-4]
mov	ecx, DWORD PTR _x\$[esp-4]
mov	edx, DWORD PTR _y\$[esp-4]
movss	xmm1, DWORD PTR [eax+4]
movss	xmm0, DWORD PTR [eax]
movss	xmm3, DWORD PTR [eax+12]
movss	xmm2, DWORD PTR [eax+8]
shufps	xmml, xmml, 0
mulps	xmm1, XMMWORD PTR [edx]
shufps	xmm2, xmm2, 0
mulps	xmm2, XMMWORD PTR [ecx]
shufps	xmm0, xmm0, 0
mulps	xmm0, XMMWORD PTR [ecx]
shufps	xmm3, xmm3, 0
mulps	xmm3, XMMWORD PTR [edx]
addps	xmm1, xmm0
addps	xmm2, xmm3
movaps	XMMWORD PTR [ecx], xmm1
movaps	XMMWORD PTR [edx], xmm2
ret	0

Quick excourse:

• Don't be intimidated by "scary looking" ASM code

- Don't be intimidated by "scary looking" ASM code
 - Might look overwhelming first but try to get past the first feeling of overwhelmingness

- Don't be intimidated by "scary looking" ASM code
 - Might look overwhelming first but try to get past the first feeling of overwhelmingness
- Use godbolt compiler explorer to get a better idea of how your code maps to ASM instructions

- Don't be intimidated by "scary looking" ASM code
 - Might look overwhelming first but try to get past the first feeling of overwhelmingness
- Use godbolt compiler explorer to get a better idea of how your code maps to ASM instructions
 - It even comes with documentation for ASM instructions if you hover over them in godbolt

- Don't be intimidated by "scary looking" ASM code
 - Might look overwhelming first but try to get past the first feeling of overwhelmingness
- Use godbolt compiler explorer to get a better idea of how your code maps to ASM instructions
 - It even comes with documentation for ASM instructions if you hover over them in godbolt
- Play with different compiler options to see how this affects ASM code generation

Quick excourse:

- Don't be intimidated by "scary looking" ASM code
 - Might look overwhelming first but try to get past the first feeling of overwhelmingness
- Use godbolt compiler explorer to get a better idea of how your code maps to ASM instructions
 - It even comes with documentation for ASM instructions if you hover over them in godbolt
- Play with different compiler options to see how this affects ASM code generation

https://godbolt.org/

• If you want to make *sure* your code gets vectorized, do it yourself

- If you want to make *sure* your code gets vectorized, do it yourself
- Which instruction set? (Easy for ARM, since there's only NEON)

- If you want to make *sure* your code gets vectorized, do it yourself
- Which instruction set? (Easy for ARM, since there's only NEON)
 - \circ MMX (jk, this is ancient)

- If you want to make *sure* your code gets vectorized, do it yourself
- Which instruction set? (Easy for ARM, since there's only NEON)
 - MMX (jk, this is ancient)
 - SSE2

- If you want to make *sure* your code gets vectorized, do it yourself
- Which instruction set? (Easy for ARM, since there's only NEON)
 - MMX (jk, this is ancient)
 - SSE2
 - SSSE3

- If you want to make *sure* your code gets vectorized, do it yourself
- Which instruction set? (Easy for ARM, since there's only NEON)
 - \circ MMX (jk, this is ancient)
 - SSE2
 - SSSE3
 - o SSE4

- If you want to make *sure* your code gets vectorized, do it yourself
- Which instruction set? (Easy for ARM, since there's only NEON)
 - MMX (jk, this is ancient)
 - o SSE2
 - SSSE3
 - o SSE4
 - o AVX

- If you want to make *sure* your code gets vectorized, do it yourself
- Which instruction set? (Easy for ARM, since there's only NEON)
 - \circ MMX (jk, this is ancient)
 - o SSE2
 - o SSSE3
 - o SSE4
 - o AVX
 - AVX-512

- If you want to make *sure* your code gets vectorized, do it yourself
- Which instruction set? (Easy for ARM, since there's only NEON)
 - MMX (jk, this is ancient)
 - o SSE2
 - SSSE3
 - o SSE4
 - o AVX
 - AVX-512
- Decision is dependent on support of target hardware
 - E.g: AVX-512 support is very limited

- If you want to make *sure* your code gets vectorized, do it yourself
- Which instruction set? (Easy for ARM, since there's only NEON)
 - MMX (jk, this is ancient)
 - o SSE2
 - SSSE3
 - o SSE4
 - o AVX
 - AVX-512
- Decision is dependent on support of target hardware
 - E.g: AVX-512 support is very limited
- Rule of thumb: Steam hardware survey
 - <u>https://store.steampowered.com/hwsurvey/Steam-Hardware-</u>
 <u>Software-Survey-Welcome-to-Steam</u>

SSE2	100.00%	0.00%
SSE3	100.00%	0.00%
SSSE3	99.56%	+0.05%
SSE4.1	99.32%	+0.07%
SSE4.2	99.08%	+0.09%
AVX	95.46%	+0.44%
AVX2	89.77%	+0.93%
SSE4a	32.29%	+0.02%
AVX512CD	9.55%	-0.03%
AVX512F	9.55%	-0.03%
AVX512VNNI	9.46%	-0.01%
AVX512ER	0.00%	0.00%
AVX512PF	0.00%	0.00%

SSE2	100.00%	0.00%
SSE3	100.00%	0.00%
SSSE3	99.56%	+0.05%
SSE4.1	99.32%	+0.07%
SSE4.2	99.08%	+0.09%
AVX	95.46%	+0.44%
AVX2	89.77%	+0.93%
SSE4a	32.29%	+0.02%
AVX512CD	9.55%	-0.03%
AVX512F	9.55%	-0.03%
AVX512VNNI	9.46%	-0.01%
AVX512ER	0.00%	0.00%
AVX512PF	0.00%	0.00%

- SSE2&SSE3 safe to use
 - "even a microwave has SSE2 support"

SSE2	100.00%	0.00%
SSE3	100.00%	0.00%
SSSE3	99.56%	+0.05%
SSE4.1	99.32%	+0.07%
SSE4.2	99.08%	+0.09%
AVX	95.46%	+0.44%
AVX2	89.77%	+0.93%
SSE4a	32.29%	+0.02%
AVX512CD	9.55%	-0.03%
AVX512F	9.55%	-0.03%
AVX512VNNI	9.46%	-0.01%
AVX512ER	0.00%	0.00%
AVX512PF	0.00%	0.00%

- SSE2&SSE3 safe to use
 - \circ "even a microwave has SSE2 support"
- AVX2 tempting but have to check for support

SSE2	100.00%	0.00%
SSE3	100.00%	0.00%
SSSE3	99.56%	+0.05%
SSE4.1	99.32%	+0.07%
SSE4.2	99.08%	+0.09%
AVX	95.46%	+0.44%
AVX2	89.77%	+0.93%
SSE4a	32.29%	+0.02%
AVX512CD	9.55%	-0.03%
AVX512F	9.55%	-0.03%
AVX512VNNI	9.46%	-0.01%
AVX512ER	0.00%	0.00%
AVX512PF	0.00%	0.00%

• SSE2&SSE3 safe to use

- \circ "even a microwave has SSE2 support"
- AVX2 tempting but have to check for support
 - Support has to be checked at runtime using CPUID

SSE2	100.00%	0.00%
SSE3	100.00%	0.00%
SSSE3	99.56%	+0.05%
SSE4.1	99.32%	+0.07%
SSE4.2	99.08%	+0.09%
AVX	95.46%	+0.44%
AVX2	89.77%	+0.93%
SSE4a	32.29%	+0.02%
AVX512CD	9.55%	-0.03%
AVX512F	9.55%	-0.03%
AVX512VNNI	9.46%	-0.01%
AVX512ER	0.00%	0.00%
AVX512PF	0.00%	0.00%

• SSE2&SSE3 safe to use

- "even a microwave has SSE2 support"
- AVX2 tempting but have to check for support
 - \circ ~ Support has to be checked at runtime using CPUID ~
 - Intel[®] 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference Table 3-8

SSE2	100.00%	0.00%
SSE3	100.00%	0.00%
SSSE3	99.56%	+0.05%
SSE4.1	99.32%	+0.07%
SSE4.2	99.08%	+0.09%
AVX.	95.46%	+0.44%
AVX2	89.77%	+0.93%
SSE4a	32.29%	+0.02%
AVX512CD	9.55%	-0.03%
AVX512F	9.55%	-0.03%
AVX512VNNI	9.46%	-0.01%
AVX512ER	0.00%	0.00%
AVX512PF	0.00%	0.00%

• SSE2&SSE3 safe to use

- "even a microwave has SSE2 support"
- AVX2 tempting but have to check for support
 - \circ ~ Support has to be checked at runtime using CPUID
 - Intel[®] 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference Table 3-8

int info[4]; __cpuid(info, 0x07);

- if(info[1] & (1<<5)){
 printf("AVX2 support!");</pre>
- } else {

printf("No AVX2 support!");

SSE2	100.00%	0.00%
SSE3	100.00%	0.00%
SSSE3	99.56%	+0.05%
SSE4.1	99.32%	+0.07%
SSE4.2	99.08%	+0.09%
AVX	95.46%	+0.44%
AVX2	89.77%	+0.93%
SSE4a	32.29%	+0.02%
AVX512CD	9.55%	-0.03%
AVX512F	9.55%	-0.03%
AVX512VNNI	9.46%	-0.01%
AVX512ER	0.00%	0.00%
AVX512PF	0.00%	0.00%

• SSE2&SSE3 safe to use

- "even a microwave has SSE2 support"
- AVX2 tempting but have to check for support
 - \circ ~ Support has to be checked at runtime using CPUID
 - Intel[®] 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference Table 3-8

int info[4];

___cpuid(info, **0x07**);

if(info[1] & (1<<5)){

printf("AVX2 support!");

} else {

printf("No AVX2 support!");

https://learn.microsoft.com/enus/cpp/intrinsics/cpuid-cpuidex

- Problem lends itself to be processes by SIMD instructions
 - 4 Pixel im parallel with SSE, 8 with AVX and 16 with AVX-512

- Problem lends itself to be processes by SIMD instructions
 - 4 Pixel im parallel with SSE, 8 with AVX and 16 with AVX-512
- Code too long to make sense to show here in detail, but for good measure:

- Problem lends itself to be processes by SIMD instructions
 4 Pixel im parallel with SSE, 8 with AVX and 16 with AVX-512
- Code too long to make sense to show here in detail, but for good measure:

Constitution of the second se Second seco

- Problem lends itself to be processes by SIMD instructions \bullet • 4 Pixel im parallel with SSE, 8 with AVX and 16 with AVX-512
- Code too long to make sense to show here in detail, but for good measure:

All sets a set large all a reactions are a set of the s

Filling Sectors - And Statistical International Conference on Confere

• Code changes necessary:

```
void WorkerMain(SharedWorkerData* sharedData){
while(true) {
  RotateJobData* jobData;
  if(sharedData->jobLock.lock()){
      if(sharedData->jobCount == 0)
          return;
      jobData = &sharedData[sharedData->jobCount--];
      sharedData->jobLock.unlock();
  }
  if(AVX2SupportDetected()) //Check for AVX2 support using CPUID
```

RotateImageBlockAVX2(jobData->startX, jobData->startY, jobData->outputImage, jobData->inputImage, jobData->rotateTransform);

else

RotateImageBlock(jobData->startX, jobData->startY,jobData->outputImage, jobData->inputImage, jobData->rotateTransform);

What does VTune say?

What does VTune say?

	»	
Function / Call Stack	Vectorization	
Rotate	100.0%	

What does VTune say?

Function / Call Stack	Vectorization
Rotate	100.0%

Even tells us what instruction sets have

been used:

Eurotion / Call Stack	Vectorization			
Fundation F Calli Station	% of FP Ops	FP Ops: Packed	FP Ops: Scalar	Vector Instruction Set
Rotate	4.7%	100.0%	0.0%	AVX(128); AVX(256); AVX2(256); AVX2GATHER(256); FMA(256)

What does VTune say?

Function / Call Stack	Vectorization
Rotate	100.0%

Even tells us what instruction sets have

been used:

Europian / Call Stock	Vectorization			
Purchair Call Stack	% of FP Ops	FP Ops: Packed	FP Ops: Scalar	Vector Instruction Set
Rotate	4.7%	100.0%	0.0%	AVX(128); AVX(256); AVX2(256); AVX2GATHER(256); FMA(256)

Rotation by 22.5 degrees took 10.7ms
What does VTune say?

Function / Call Stack	Vectorization
Rotate	100.0%

Even tells us what instruction sets have

been used:

Eurotion / Call Stack	Vectorization			
Punction / Gali Stack	% of FP Ops	FP Ops: Packed	FP Ops: Scalar	Vector Instruction Set
Rotate	4.7%	100.0%	0.0%	AVX(128); AVX(256); AVX2(256); AVX2GATHER(256); FMA(256)

• Most invasive code change

- Most invasive code change
- Scalar code path still needed (in case hardware architecture doesn't support AVX2)

- Most invasive code change
- Scalar code path still needed (in case hardware architecture doesn't support AVX2)
- Set of people who can debug and read this code has been reduced significantly

- Most invasive code change
- Scalar code path still needed (in case hardware architecture doesn't support AVX2)
- Set of people who can debug and read this code has been reduced significantly
- Nice tradeoff between SIMD speed & code readability: Intel ISPC <u>https://ispc.github.io/</u> (Compiler build around code vectorization)

- Most invasive code change
- Scalar code path still needed (in case hardware architecture doesn't support AVX2)
- Set of people who can debug and read this code has been reduced significantly
- Nice tradeoff between SIMD speed & code readability: Intel ISPC <u>https://ispc.github.io/</u> (Compiler build around code vectorization)
- Intel® Intrinsics Guide (SSE Instruction set overview) <u>https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html</u>

We could do more:

We could do more:

• Add AVX-512 path on supported hardware

We could do more:

- Add AVX-512 path on supported hardware
- Let each worker write into its own bitmap and merge later

We could do more:

- Add AVX-512 path on supported hardware
- Let each worker write into its own bitmap and merge later
- Manually prefetch next samples using PrefetchCacheLine() (_mm_prefetch())
 - <u>https://learn.microsoft.com/en-us/windows/win32/api/winnt/nf-winnt-prefetchcacheline</u>

We could do more:

- Add AVX-512 path on supported hardware
- Let each worker write into its own bitmap and merge later
- Manually prefetch next samples using PrefetchCacheLine() (_mm_prefetch())
 - <u>https://learn.microsoft.com/en-us/windows/win32/api/winnt/nf-winnt-prefetchcacheline</u>

But: Also important to know when to stop - All of the above introduces more complexity & more code - which has the potential of introducing more bugs and worse maintenance.

Result



• Never assume, always measure

- Never assume, always measure
 - Data might even be very different between CPUs (instruction have been implemented differently or even emulated)

- Never assume, always measure
 - Data might even be very different between CPUs (instruction have been implemented differently or even emulated)
 - PDEP as an example of an instruction with vastly different performance metrics between different CPUs

- Never assume, always measure
 - Data might even be very different between CPUs (instruction have been implemented differently or even emulated)
 - PDEP as an example of an instruction with vastly different performance metrics between different CPUs

Zen3 Updates (2) Integer Instructions						
AnandTech	Instruction	Zen2	Zen 3			
PDEP/PEXT	Parallel Bits Deposit/Extreact	300 cycle latency 250 cycles per 1	3 cycle latency 1 per clock			

https://www.anandtech.com/show/16214/amd-zen-3-ryzen-deep-dive-review-5950x-5900x-5800x-and-5700x-tested/6

- Know your target hardware (RTFM)
 - x86_64 & ARM aren't going away any time soon

- Know your target hardware (RTFM)
 - $\circ \quad x86_64 \ \& ARM$ aren't going away any time soon
- Make yourself familiar with vendor specific profilers

- Know your target hardware (RTFM)
 - $\circ \quad x86_64 \ \& ARM$ aren't going away any time soon
- Make yourself familiar with vendor specific profilers
 - Only for low-level optimization though. For high-level stuff I'd use Superluminal or other sample-based profilers.

- Know your target hardware (RTFM)
 - x86_64 & ARM aren't going away any time soon
- Make yourself familiar with vendor specific profilers
 - Only for low-level optimization though. For high-level stuff I'd use Superluminal or other sample-based profilers.
- Verify your assumptions regarding compiler optimizations

- Know your target hardware (RTFM)
 - x86_64 & ARM aren't going away any time soon
- Make yourself familiar with vendor specific profilers
 - Only for low-level optimization though. For high-level stuff I'd use Superluminal or other sample-based profilers.
- Verify your assumptions regarding compiler optimizations
- SIMD might not always be the best first choice

- Know your target hardware (RTFM)
 - $\circ \quad$ x86_64 & ARM aren't going away any time soon
- Make yourself familiar with vendor specific profilers
 - Only for low-level optimization though. For high-level stuff I'd use Superluminal or other sample-based profilers.
- Verify your assumptions regarding compiler optimizations
- SIMD might not always be the best first choice
- Know when to stop (Ideally you'd know your performance budget)

Where to go from here?

- Mike Acton "Data-Oriented Design and C++" <u>https://www.youtube.com/watch?v=rX0ItVEVjHc</u>
- Casey Muratori "'Clean Code', Horrible Performance" https://www.youtube.com/watch?v=tD5NrevFtbU&t=1s
- Jon Blow "Preventing the Collapse of Civilization" <u>https://www.youtube.com/watch?v=q3OCFfDStgM</u>
- Ulrich Drepper "What every programmer should know about memory" <u>https://people.freebsd.org/~lstewart/articles/cpumemory.pdf</u>
- John L. Hennessy, David A. Patterson "Computer Architecture" <u>https://www.oreilly.com/library/view/computer-architecture-5th/9780123838735/</u>
- Scott Meyers "CPU Cache and why you care" <u>https://www.youtube.com/watch?v=WDIkqP4JbkE</u>

Thanks for your attention!

Reach out in case of questions!

🥑 @FelixK_15

@FelixK15 (gamedev.place)

in Felix Klinge

felix [at] k15tech [dot] com