A Relationship Guide For Your Hardware

...

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?”

Why should I care?

“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”

Why should I care?

“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”

Why should I care?

“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

Why should I care?

“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

Why should I care?

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

http://www.youtube.com/watch?v=CT7nnXej2K4

Example: 2D bitmap rotation

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

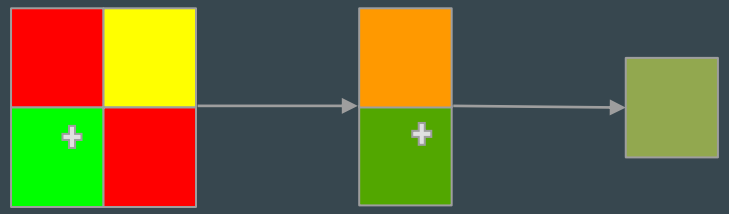
- Rotating a 2D image on the CPU with bilinear sampling
- Image is 4096x4096

Example: 2D bitmap rotation

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:



Example: 2D bitmap rotation

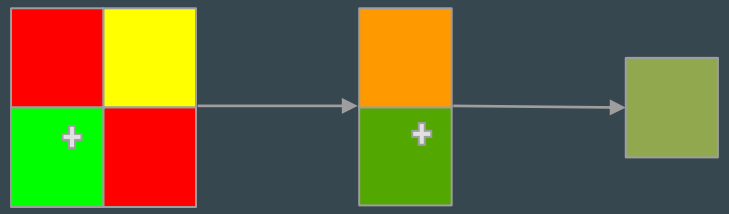
Let's work with a concrete example during this talk:

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

Input:

- Unrotated 2D image
- Transformation matrix

Bilinear Sampling:



Example: 2D bitmap rotation

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

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

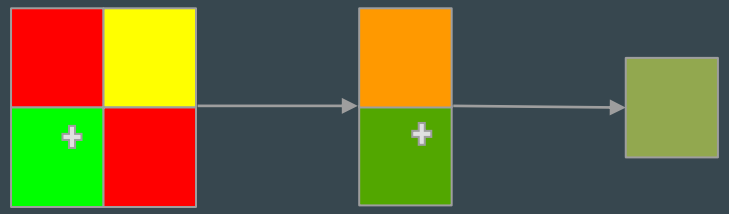
Input:

- Unrotated 2D image
- Transformation matrix

Output:

- Rotated 2D image

Bilinear Sampling:



Example: 2D bitmap rotation

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

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

Input:

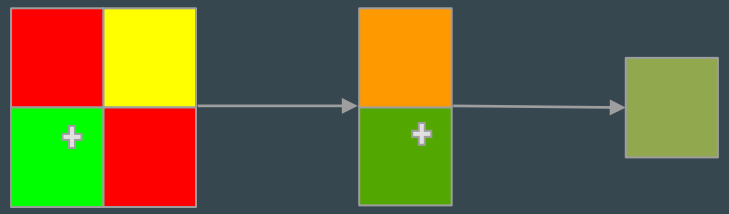
- Unrotated 2D image
- Transformation matrix

Output:

- Rotated 2D image

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

Bilinear Sampling:



Example: 2D bitmap rotation

Basically a 2D matrix transformation: $\begin{Bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{Bmatrix}$

- 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

Example: 2D bitmap rotation

- 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 %.1f degree took %.1fms\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);
  }
  }
}
```

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 how the hardware can be optimized for?

How do we know what we can improve on without knowing the hardware?

-> Documentation (RTFM)

AMD: https://gpuopen.com/ryzen-performance/

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

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

How do we know how the hardware can be optimized for?

How do we know what we can improve on without knowing the hardware?

-> Documentation (RTFM)

AMD: https://gpuopen.com/ryzen-performance/

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

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

We'll focus on Intel for this talk.

How do we know how the hardware can be optimized for?

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!”.

How do we know how the hardware can be optimized for?

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.

How do we know how the hardware can be optimized for?

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.

How do we know how the hardware can be optimized for?

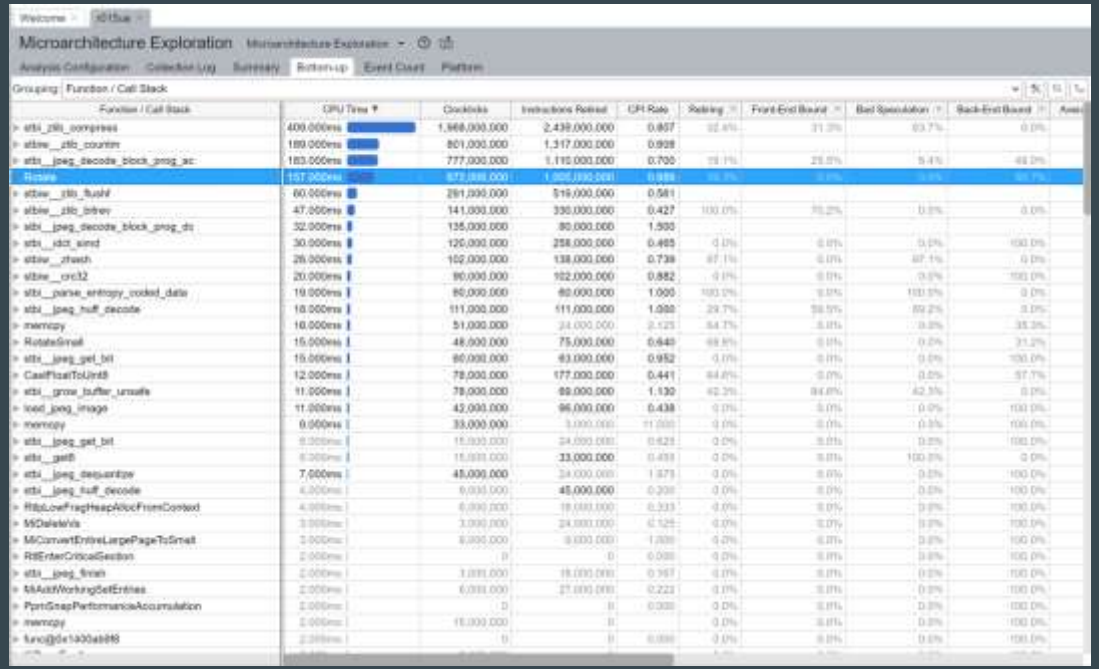
It's in the CPU vendor's best interest not to have headlines like "My game runs slower on [CPU from vendor A] than on [CPU from vendor B] but the CPUs are the same speed!".

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
  - AMD uProf
  - Qualcomm Snapdragon Profiler

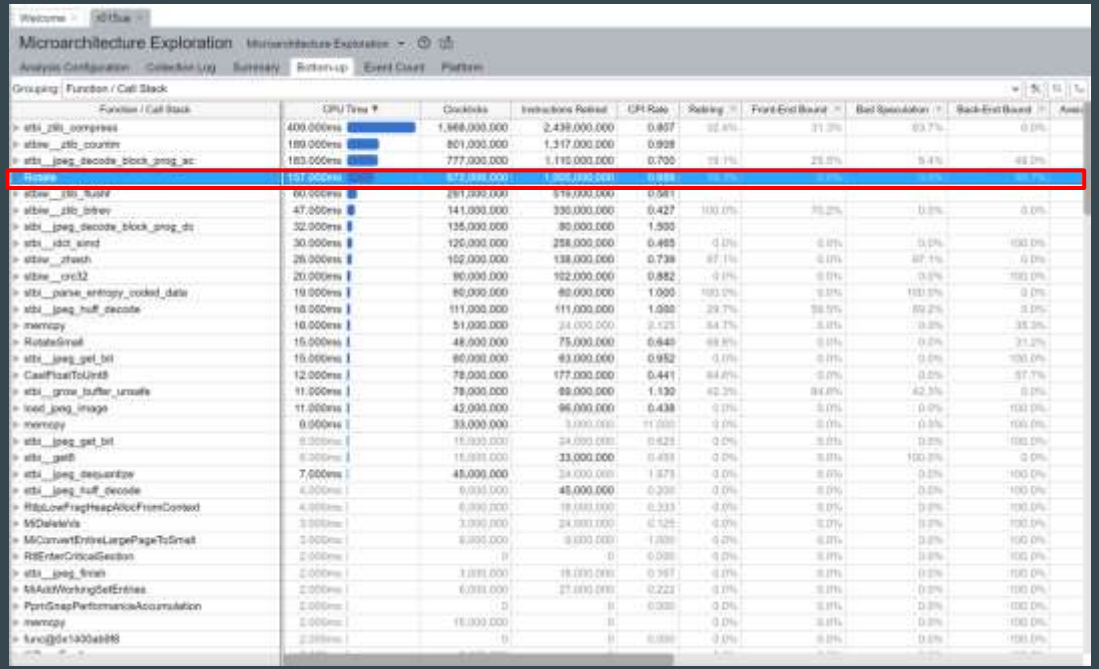
Optimization 1: Better Execution Unit Utilization

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



Optimization 1: Better Execution Unit Utilization

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



Optimization 1: Better Execution Unit Utilization

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

Optimization 1: Better Execution Unit Utilization

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

Strong indication of sub-optimal execution unit utilization

Optimization 1: Better Execution Unit Utilization

Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation	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.

Optimization 1: Better Execution Unit Utilization

Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation	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.

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

Optimization 1: Better Execution Unit Utilization

Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation	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.

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

- Front-End: Transforms ASM into u-Ops

Optimization 1: Better Execution Unit Utilization

Function / Call Stack	CPI Rate	Front-End Bound	Bad Speculation	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.

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

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

Optimization 1: Better Execution Unit Utilization

What are Execution Units?

Optimization 1: Better Execution Unit Utilization

What are Execution Units?

- Most (all?) modern CPUs are superscalar CPUs.

Optimization 1: Better Execution Unit Utilization

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).

Optimization 1: Better Execution Unit Utilization

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.

Optimization 1: Better Execution Unit Utilization

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

Optimization 1: Better Execution Unit Utilization

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

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

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					

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.

Optimization 1: Better Execution Unit Utilization

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

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

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					

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

Optimization 1: Better Execution Unit Utilization

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);
    }
  }
}
```

Optimization 1: Better Execution Unit Utilization

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);
    }
  }
}
```

Full of dependencies :(

Optimization 1: Better Execution Unit Utilization

Loop unrolling to the rescue!

Optimization 1: Better Execution Unit Utilization

Loop unrolling to the rescue!

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

Optimization 1: Better Execution Unit Utilization

Loop unrolling to the rescue!

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

Optimization 1: Better Execution Unit Utilization

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

Optimization 1: Better Execution Unit Utilization

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);
  }
  }
}
```

Optimization 1: 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%

Optimization 1: 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

Optimization 1: 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

Optimization 1: 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
- Still readable

```
Rotation by 22.5 degrees took 209.0ms
```

Optimization 1: 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
- Still readable
- Only minimal changes needed

```
Rotation by 22.5 degrees took 151.9ms
```

Optimization 2: Loop Blocking

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

Optimization 2: Loop Blocking

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?

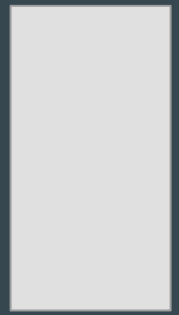
Optimization 2: Loop Blocking

Excourse CPU Caches (High level overview):

Optimization 2: Loop Blocking

Excourse CPU Caches (High level overview):

CPU Cache

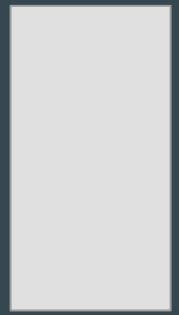


KBytes/MBytes

Optimization 2: Loop Blocking

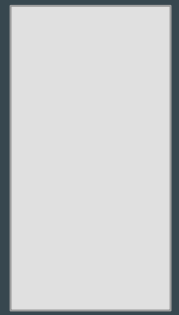
Excourse CPU Caches (High level overview):

CPU Cache



KBytes/MBytes

Main Memory

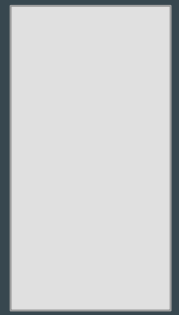


GBytes

Optimization 2: Loop Blocking

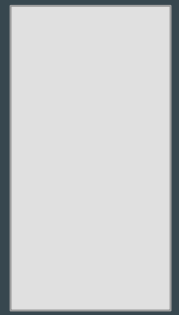
Excourse CPU Caches (High level overview):

CPU Cache



KBytes/MBytes

Main Memory



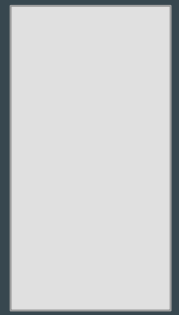
GBytes

This image has been loaded into memory: 

Optimization 2: Loop Blocking

Excourse CPU Caches (High level overview):

CPU Cache



KBytes/MBytes

Main Memory

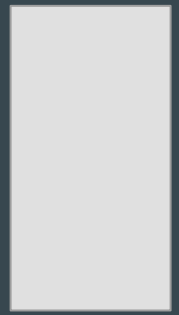
GBytes

This image has been loaded into memory: 

Optimization 2: Loop Blocking

Excourse CPU Caches (High level overview):

CPU Cache



KBytes/MBytes

Main Memory

GBytes

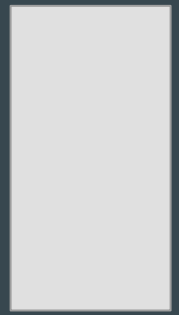
Assume we want to access one pixel after another

This image has been loaded into memory:

Optimization 2: Loop Blocking

Excourse CPU Caches (High level overview):

CPU Cache



KBytes/MBytes

Main Memory

GBytes

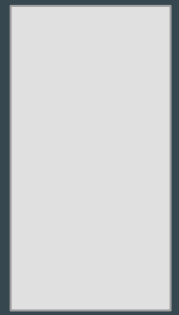
Assume we want to access one pixel after another

This image has been loaded into memory:

Optimization 2: Loop Blocking

Excourse CPU Caches (High level overview):

CPU Cache



KBytes/MBytes

Main Memory

GBytes

Assume we want to access one pixel after another

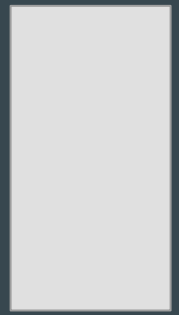
For each access, the CPU first checks the cache

This image has been loaded into memory:

Optimization 2: Loop Blocking

Excourse CPU Caches (High level overview):

CPU Cache



KBytes/MBytes

Main Memory

GBytes

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.

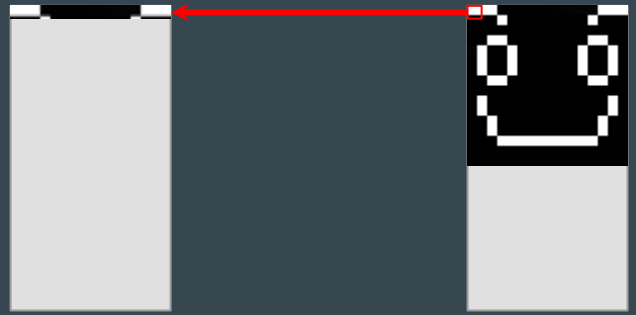
This image has been loaded into memory:

Optimization 2: Loop Blocking

Excourse CPU Caches (High level overview):

CPU Cache

Main Memory



KBytes/MBytes

GBytes

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.

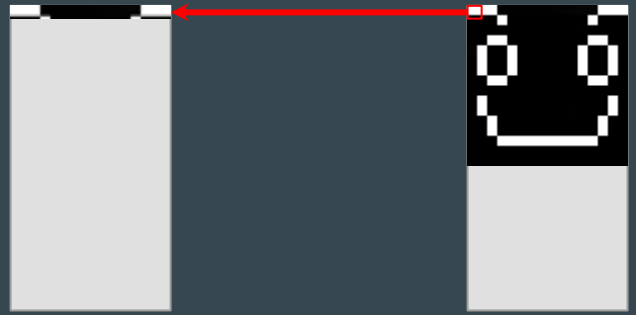
This image has been loaded into memory:

Optimization 2: Loop Blocking

Excourse CPU Caches (High level overview):

CPU Cache

Main Memory



KBytes/MBytes

GBytes

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:

Optimization 2: Loop Blocking

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

Optimization 2: Loop Blocking

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)

Optimization 2: Loop Blocking

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

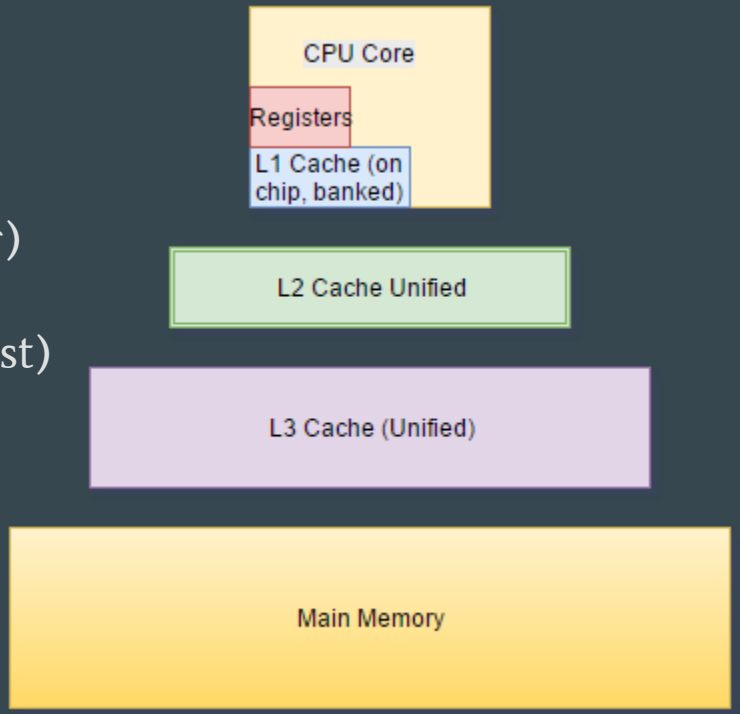
This is done because it is assumed that you're also interested in neighboring data and not just one byte (or pixel in this case)

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

Optimization 2: Loop Blocking

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)



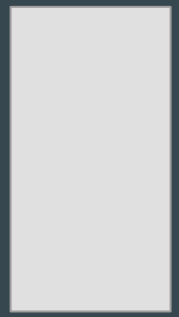
https://en.wikipedia.org/wiki/Cache_hierarchy

Optimization 2: Loop Blocking

- Cache Miss
- Cache Hit

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

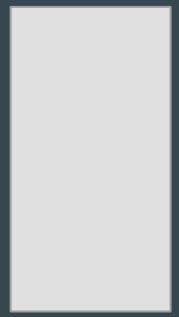
GBytes

Cache Miss

Cache Hit

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

GBytes

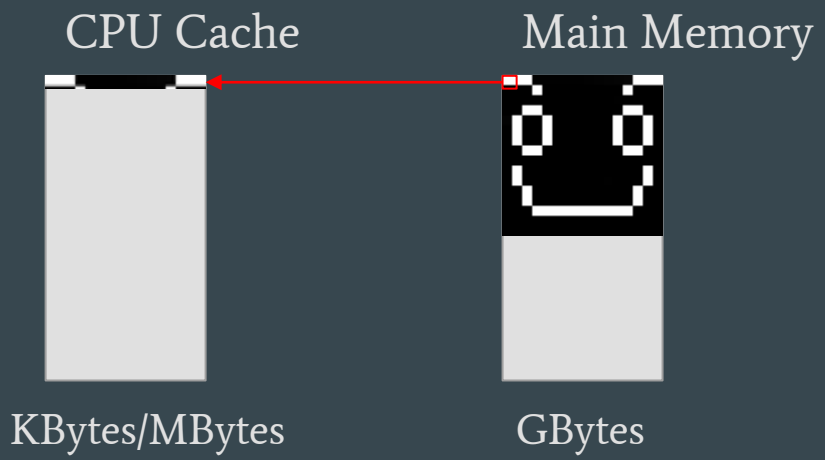
Cache Miss

Cache Hit

Assume code like this:

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

Optimization 2: Loop Blocking



Cache Miss

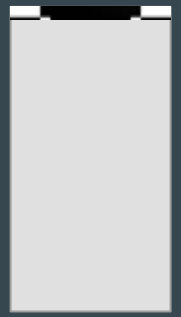
Cache Hit

Assume code like this:

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

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

GBytes

Cache Miss

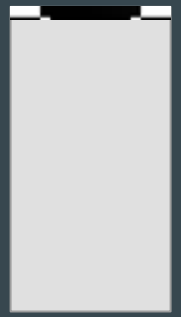
Cache Hit

Assume code like this:

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

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

GBytes

Cache Miss

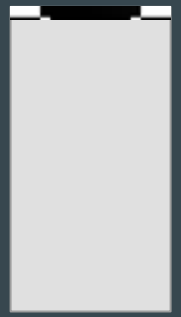
Cache Hit

Assume code like this:

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

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

GBytes

Cache Miss

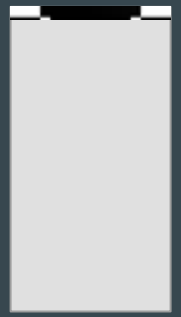
Cache Hit

Assume code like this:

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

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

GBytes

Cache Miss

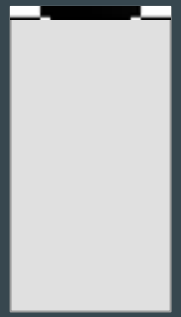
Cache Hit

Assume code like this:

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

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

GBytes

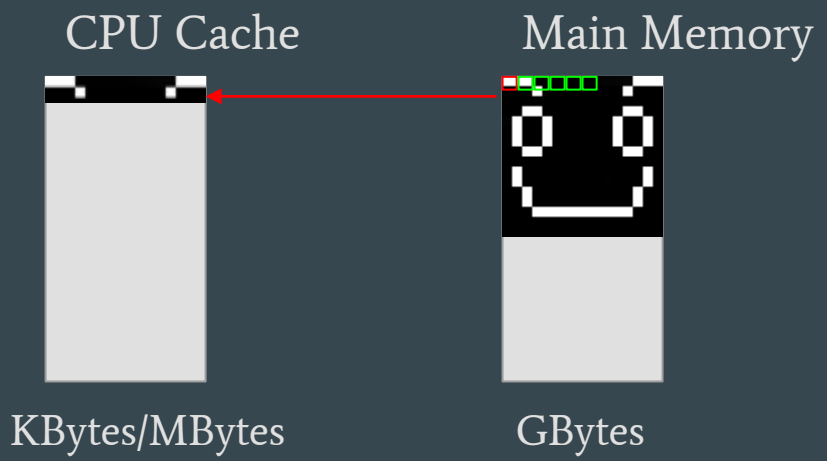
Cache Miss

Cache Hit

Assume code like this:

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

Optimization 2: Loop Blocking



Cache Miss

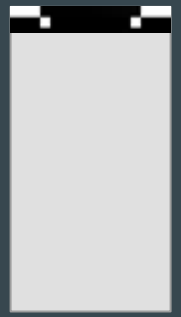
Cache Hit

Assume code like this:

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

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

GBytes

Cache Miss

Cache Hit

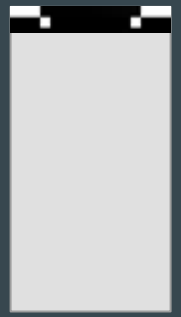
Assume code like this:

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

Prefetcher fetches next cache line because of the sequential access pattern

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

GBytes

Cache Miss

Cache Hit

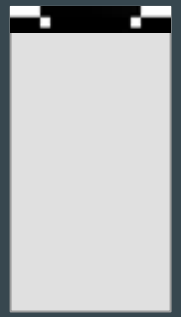
Assume code like this:

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

Prefetcher fetches next cache line because of the sequential access pattern

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

GBytes

Cache Miss

Cache Hit

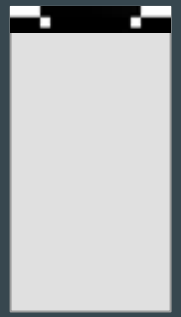
Assume code like this:

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

Prefetcher fetches next cache line because of the sequential access pattern

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

GBytes

Cache Miss

Cache Hit

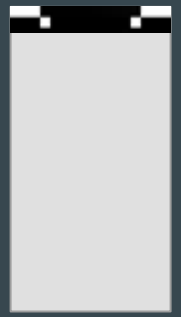
Assume code like this:

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

Prefetcher fetches next cache line because of the sequential access pattern

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

GBytes

Cache Miss

Cache Hit

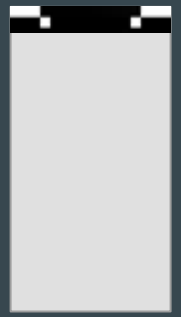
Assume code like this:

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

Prefetcher fetches next cache line because of the sequential access pattern

Optimization 2: Loop Blocking

CPU Cache



KBytes/MBytes

Main Memory

GBytes

Cache Miss

Cache Hit

Assume code like this:

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

Prefetcher fetches next cache line because of the sequential access pattern

Optimization 2: Loop Blocking

Let’s revisit the algorithm:

Optimization 2: Loop Blocking

Let's revisit the algorithm:

We want to rotate this image by 50°:

Optimization 2: Loop Blocking

Let's revisit the algorithm:

We want to rotate this image by 50°:

Optimization 2: Loop Blocking

Let's revisit the algorithm:

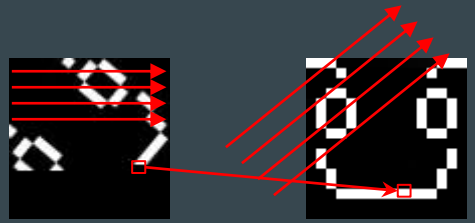
We want to rotate this image by 50°:

- Write access is sequential, no problem here

Optimization 2: Loop Blocking

Let's revisit the algorithm:

We want to rotate this image by 50°:

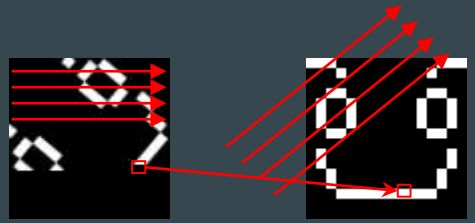


- Write access is sequential, no problem here
- Read access is non-sequential
  - Worst case: every read is a cache miss

Optimization 2: Loop Blocking

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
  - Worst case: every read is a cache miss

What can we do about it?

Optimization 2: Loop Blocking

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

Optimization 2: Loop Blocking

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

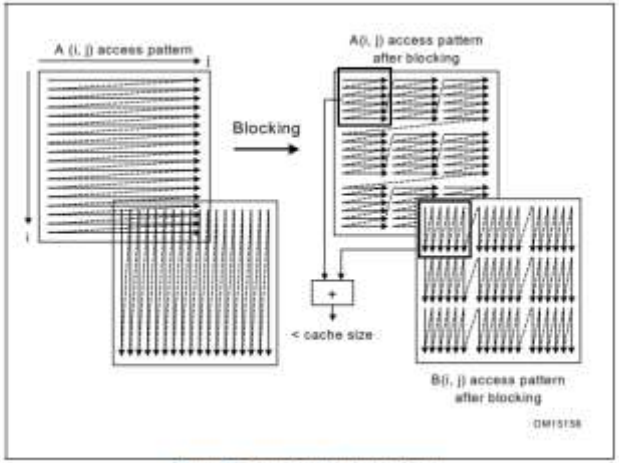


Figure 5-5. Loop Blocking Access Pattern

Optimization 2: Loop Blocking

```
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 ) {
  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);
  }
  }
}
```

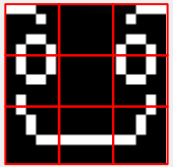
Optimization 2: Loop Blocking

```
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 ) {
  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);
  }
  }
}
```

Optimization 2: Loop Blocking

```
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 ) {
  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);
  }
  }
}
```

Optimization 2: Loop Blocking

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

Optimization 2: Loop Blocking

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

Optimization 2: Loop Blocking

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

Optimization 2: Loop Blocking

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

Optimization 2: Loop Blocking

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
```

Optimization 2: Loop Blocking

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 123.9ms
```

Optimization 3: Multithreading

- So far we only used one core

Optimization 3: Multithreading

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

Optimization 3: Multithreading

- 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

Optimization 3: Multithreading

- 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

Optimization 3: Multithreading

- 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)

Optimization 3: Multithreading

Optimization 3: Multithreading

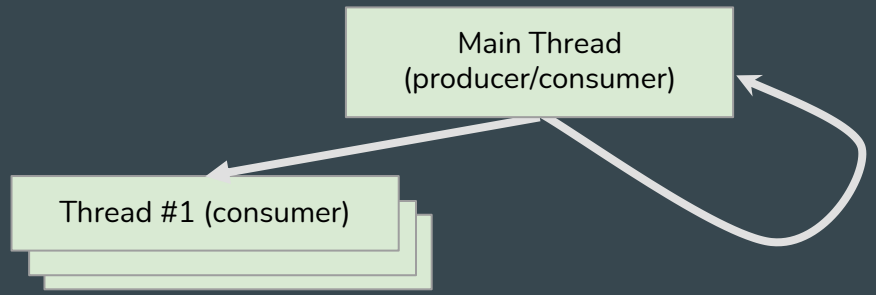
- Job system lends itself perfectly for this use case
  - General idea: break work down into independent jobs, assign threads as workers, each worker works on one job

Optimization 3: Multithreading

- 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

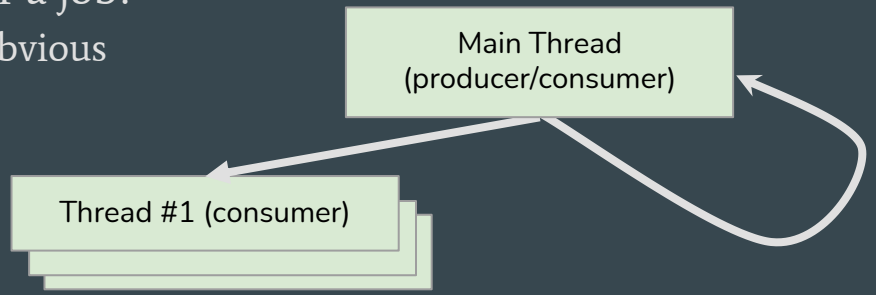
Optimization 3: Multithreading

- 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



Optimization 3: Multithreading

- 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



Optimization 3: Multithreading

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

Optimization 3: Multithreading

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
  - GetLogicalProcessorInformation() for win32
  - get_nprocs() for posix

```
std::thread** CreateWorker(SharedWorkerData* workerData){
  unsigned int workerCount = std::thread::hardware_concurrency()-1u;
  std::thread** worker = new std::thread*[workerCount];
  for(int i = 0; i < workerCount; ++i){
    worker[i] = new std::thread(&WorkerMain, workerData);
  }
  return worker;
}
```

Optimization 3: Multithreading

Create independent jobs

- Group job data into new data structure

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

Optimization 3: Multithreading

- 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 ){
      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;
}
```

Optimization 3: Multithreading

- 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);
  }
}
```

Optimization 3: Multithreading

- 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-1u; ++i){
        workers[i]->join();
    }
}
```

Optimization 3: Multithreading

- 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-1u; ++i){
        workers[i]->join();
    }
}
```

```
Rotation by 22.5 degrees took 123.9ms
```

Optimization 3: Multithreading

- 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-1u; ++i){
        workers[i]->join();
    }
}
```

```
Rotation by 22.5 degrees took 10.7ms
```

Optimization 3: Multithreading

- If you're using an existing engine or framework, job system is most likely already in place
  - Eg: Job System in Unity https://docs.unity3d.com/Manual/JobSystem.html
- 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

Optimization 3: Multithreading

- Many traps to fall into

Optimization 3: Multithreading

- Many traps to fall into
  - False sharing (Performance)
  - Race conditions (Behavior)
  - Deadlocks (Crashes)

Optimization 3: Multithreading

- 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

Optimization 3: Multithreading

- 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)

Optimization 4: SIMD

SIMD = Single Instruction Multiple Data

Optimization 4: SIMD

SIMD = Single Instruction Multiple Data

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

Optimization 4: SIMD

SIMD = Single Instruction Multiple Data

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

Example multiplying numbers:

Optimization 4: SIMD

SIMD = Single Instruction Multiple Data

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;
}
```

Optimization 4: SIMD

SIMD = Single Instruction Multiple Data

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);
}
```

Optimization 4: SIMD

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.

Optimization 4: SIMD

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

Optimization 4: SIMD

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

"The compiler will optimize it!"

Optimization 4: SIMD

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

"The compiler will optimize it!"

Optimization 4: SIMD

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

Optimization 4: SIMD

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

Optimization 4: SIMD

- 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

Optimization 4: SIMD

- 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:

Optimization 4: SIMD

- 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 ▼			Memory Bound	Vectorization
	Effective Time	Spin Time	Overhead Time		
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

Optimization 4: SIMD

- 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 ▼			Memory Bound	Vectorization
	Effective Time	Spin Time	Overhead Time		
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

Optimization 4: SIMD

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

Optimization 4: SIMD

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;
    }
}
```

Optimization 4: SIMD

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;
    }
}
```

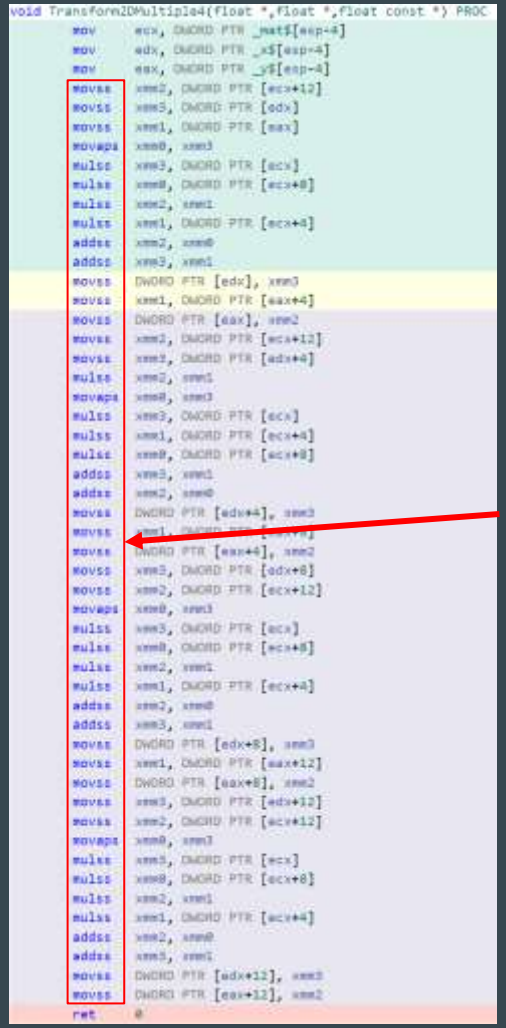
```
void Transform2DMultiple4(float *,float *,float const *) PROC
        mov     ecx, DWORD PTR _mat$[esp-4]
        mov     edx, DWORD PTR _x$[esp-4]
        mov     eax, DWORD PTR _y$[esp-4]
        movss   xmm2, DWORD PTR [ecx+12]
        movss   xmm3, DWORD PTR [edx]
        movss   xmm1, DWORD PTR [eax]
        movaps  xmm0, xmm3
        mulss   xmm3, DWORD PTR [ecx]
        mulss   xmm0, DWORD PTR [ecx+8]
        mulss   xmm2, xmm1
        mulss   xmm1, DWORD PTR [ecx+4]
        addss   xmm2, xmm0
        addss   xmm3, xmm1
        movss   DWORD PTR [edx], xmm3
        movss   xmm1, DWORD PTR [eax+4]
        movss   DWORD PTR [eax], xmm2
        movss   xmm2, DWORD PTR [ecx+12]
        movss   xmm3, DWORD PTR [edx+4]
        mulss   xmm2, xmm1
        movaps  xmm0, xmm3
        mulss   xmm3, DWORD PTR [ecx]
        mulss   xmm1, DWORD PTR [ecx+4]
        mulss   xmm0, DWORD PTR [ecx+8]
        addss   xmm3, xmm1
        addss   xmm2, xmm0
        movss   DWORD PTR [edx+4], xmm3
        movss   xmm1, DWORD PTR [eax+8]
        movss   DWORD PTR [eax+4], xmm2
        movss   xmm3, DWORD PTR [edx+8]
        movss   xmm2, DWORD PTR [ecx+12]
        movaps  xmm0, xmm3
        mulss   xmm3, DWORD PTR [ecx]
        mulss   xmm0, DWORD PTR [ecx+8]
        mulss   xmm2, xmm1
        mulss   xmm1, DWORD PTR [ecx+4]
        addss   xmm2, xmm0
        addss   xmm3, xmm1
        movss   DWORD PTR [edx+8], xmm3
        movss   xmm1, DWORD PTR [eax+12]
        movss   DWORD PTR [eax+8], xmm2
        movss   xmm3, DWORD PTR [edx+12]
        movss   xmm2, DWORD PTR [ecx+12]
        movaps  xmm0, xmm3
        mulss   xmm3, DWORD PTR [ecx]
        mulss   xmm0, DWORD PTR [ecx+8]
        mulss   xmm2, xmm1
        mulss   xmm1, DWORD PTR [ecx+4]
        addss   xmm2, xmm0
        addss   xmm3, xmm1
        movss   DWORD PTR [edx+12], xmm3
        movss   DWORD PTR [eax+12], xmm2
        ret     0
```

Optimization 4: SIMD

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;
    }
}
```



All scalar :(

Optimization 4: SIMD

```
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***)

Optimization 4: SIMD

```
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 *,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  xmm1, xmm1, 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
```

Optimization 4: SIMD

Quick excourse:

Optimization 4: SIMD

Quick excourse:

- Don’t be intimidated by “scary looking” ASM code

Optimization 4: SIMD

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

Optimization 4: SIMD

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

Optimization 4: SIMD

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

Optimization 4: SIMD

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

Optimization 4: SIMD

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/

Optimization 4: SIMD

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

Optimization 4: SIMD

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

Optimization 4: SIMD

- 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)

Optimization 4: SIMD

- 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

Optimization 4: SIMD

- 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

Optimization 4: SIMD

- 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
  - SSE4

Optimization 4: SIMD

- 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
  - SSE4
  - AVX

Optimization 4: SIMD

- 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
  - SSE4
  - AVX
  - AVX-512

Optimization 4: SIMD

- 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
  - SSE4
  - AVX
  - AVX-512
- Decision is dependent on support of target hardware
  - E.g: AVX-512 support is very limited

Optimization 4: SIMD

- 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
  - SSE4
  - 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
  - https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam

Optimization 4: SIMD

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%

Optimization 4: SIMD

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”

Optimization 4: SIMD

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

Optimization 4: SIMD

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
  - Support has to be checked at runtime using CPUID

Optimization 4: SIMD

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
  - 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

Optimization 4: SIMD

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
  - 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!");
}
```

Optimization 4: SIMD

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
  - 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/en-us/cpp/intrinsics/cpuid-cpuidex

Optimization 4: SIMD

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

Optimization 4: SIMD

- 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:

Optimization 4: SIMD

- 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:

Optimization 4: SIMD

- 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:

Optimization 4: SIMD

- 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);
    }
}
```

Optimization 4: SIMD

What does VTune say?

Optimization 4: SIMD

What does VTune say?

Function / Call Stack	Vectorization
Rotate	100.0%

Optimization 4: SIMD

What does VTune say?

Function / Call Stack	Vectorization
Rotate	100.0%

Even tells us what instruction sets have been used:

Function / Call Stack	Vectorization			
	% 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)

Optimization 4: SIMD

What does VTune say?

Function / Call Stack	Vectorization
Rotate	100.0%

Even tells us what instruction sets have been used:

Function / Call Stack	Vectorization			
	% 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
```

Optimization 4: SIMD

What does VTune say?

Function / Call Stack	Vectorization
Rotate	100.0%

Even tells us what instruction sets have been used:

Function / Call Stack	Vectorization			
	% 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 6.9ms
```

Optimization 4: SIMD

- Most invasive code change

Optimization 4: SIMD

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

Optimization 4: SIMD

- 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

Optimization 4: SIMD

- 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
  https://ispc.github.io/ (Compiler build around code vectorization)

Optimization 4: SIMD

- 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
  https://ispc.github.io/ (Compiler build around code vectorization)
- Intel® Intrinsics Guide (SSE Instruction set overview)
  https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

Further Work

We could do more:

Further Work

We could do more:

- Add AVX-512 path on supported hardware

Further Work

We could do more:

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

Further Work

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())
  - https://learn.microsoft.com/en-us/windows/win32/api/winnt/nf-winnt-prefetchcacheline

Further Work

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())
    - https://learn.microsoft.com/en-us/windows/win32/api/winnt/nf-winnt-prefetchcacheline

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

```
Rotation by 22.5 degrees took 209.0ms
```

Loop unrolling

```
Rotation by 22.5 degrees took 151.9ms
```

Loop Blocking

```
Rotation by 22.5 degrees took 123.9ms
```

Multithreading

```
Rotation by 22.5 degrees took 10.7ms
```

SIMD

```
Rotation by 22.5 degrees took 6.9ms
```

Conclusion

- Never assume, always measure

Conclusion

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

Conclusion

- 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

Conclusion

- 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

Conclusion

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

Conclusion

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

Conclusion

- 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.

Conclusion

- 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

Conclusion

- 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

Conclusion

- 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 when to stop (Ideally you'd know your performance budget)

Where to go from here?

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

Thanks for your attention!

Reach out in case of questions!

- @FelixK_15
- @FelixK15 (gamedev.place)
- Felix Klinge
- felix [at] k15tech [dot] com