

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * ddn;
        D, N );
    }
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * ddn);
    (Tr) R = (D * nnt - N * (ddn *
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &light);
    e.x + radiance.y + radiance.z) > 0) && (depth <
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
    random walk - done properly, closely following
    (survive)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
```

/INFOMOV/

Optimization & Vectorization

J. Bikker - Sep-Nov 2018 - Lecture 12: "Multithreading"

Welcome!



Today's Agenda:

- Introduction
- Hardware
- Trust No One / An Efficient Pattern
- Experiments
- Final Assignment



Introduction

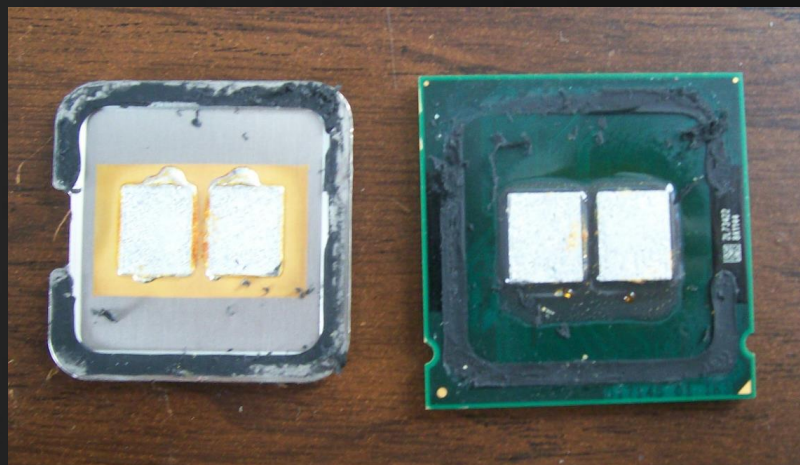
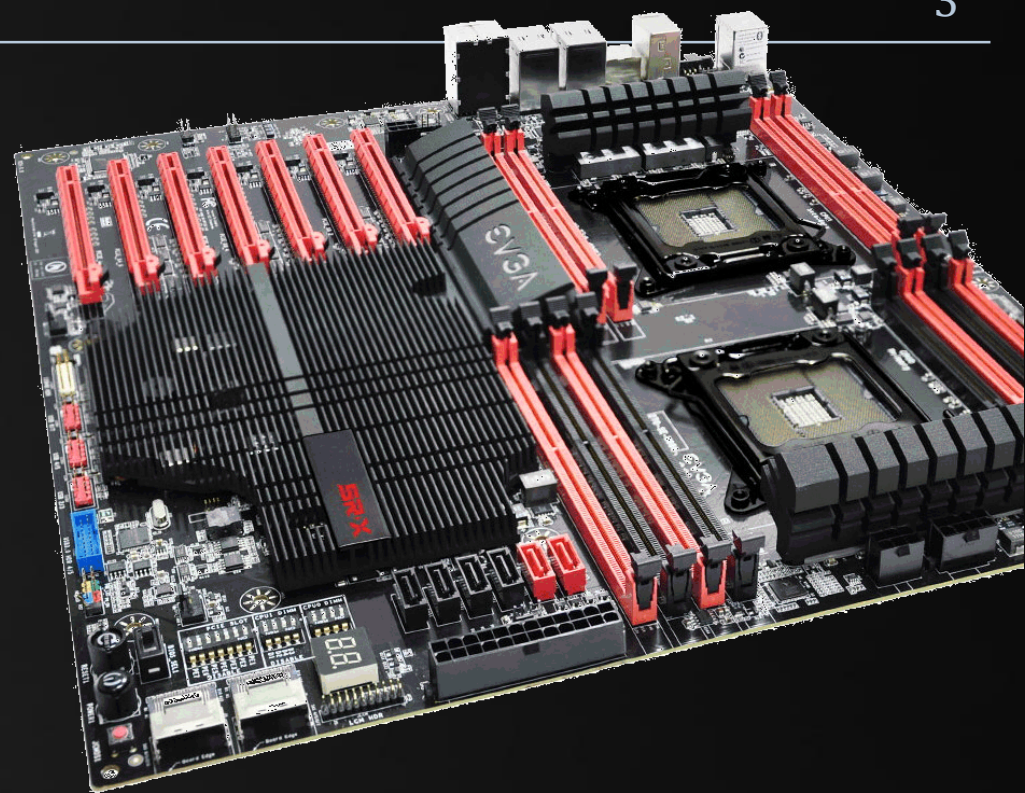
A Brief History of Many Cores

Once upon a time...

Then, in 2005: Intel’s Core 2 Duo (*April 22*).
 (Also 2005: AMD Athlon 64 X2. *April 21*.)

2007: Intel Core 2 Quad

2010: AMD Phenom II X6



```

ics
& (depth < MAXDEPTH)
    if (inside ? 1 : 0) {
        nt = nt / nc; ddn = ddn / n;
        cos2t = 1.0f - nnt;
        D, N );
        )
        at a = nt - nc, b = nt + nc;
        at Tr = 1 - (R0 + (1 - R0) *
        Tr) R = (D * nnt - N * (ddn
        E * diffuse;
        = true;
        refl + refr)) && (depth < MAXDEPTH)
        D, N );
        refl * E * diffuse;
        = true;
        MAXDEPTH)
        survive = SurvivalProbability( diffuse,
        estimation - doing it properly, closely
        if;
        radiance = SampleLight( &rand, I, &L, &light;
        e.x + radiance.y + radiance.z) > 0) && (cos2t
        w = true;
        at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
        at3 factor = diffuse * INVPI;
        at weight = Mis2( directPdf, brdfPdf );
        at cosThetaOut = dot( N, L );
        E * ((weight * cosThetaOut) / directPdf) * (radiance
        random walk - done properly, closely following 3D
        vive)
        at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf;
        survive;
        pdf;
        n = E * brdf * (dot( N, R ) / pdf);
        sion = true;
    
```



Introduction

A Brief History of Many Cores

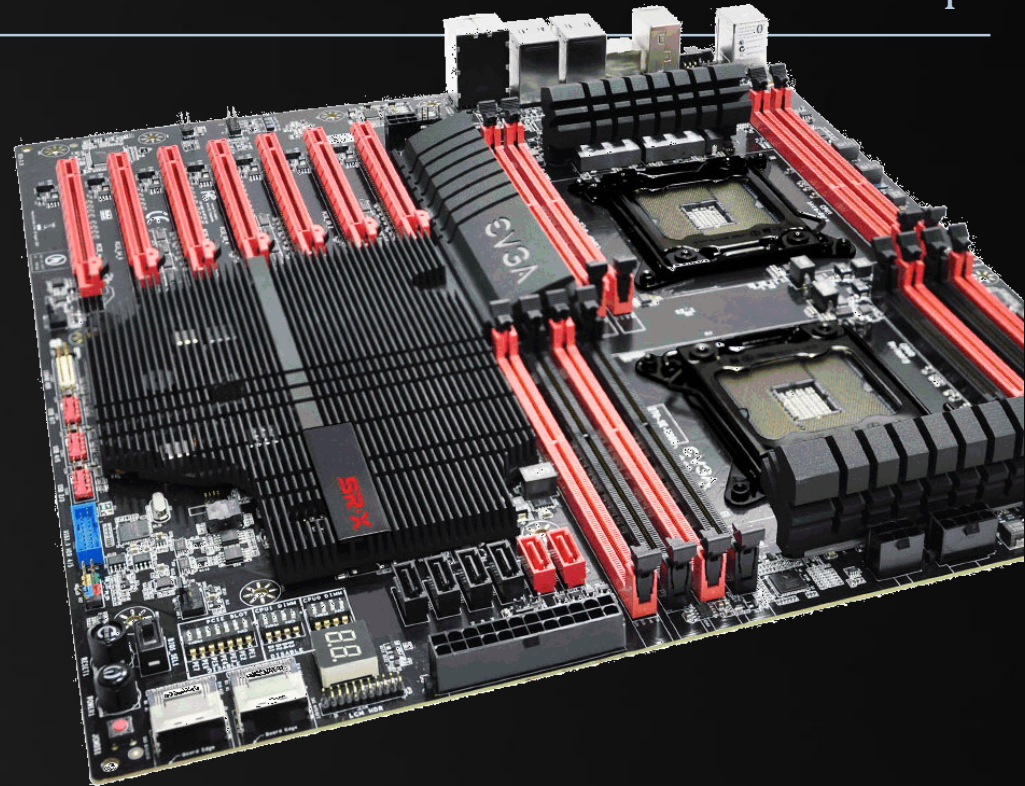
Once upon a time...

Then, in 2005: Intel's Core 2 Duo (*April 22*).
(Also 2005: AMD Athlon 64 X2. *April 21*.)

2007: Intel Core 2 Quad

2010: AMD Phenom II X6

Today...



ics
& (depth < MAXDEPTH)
= inside ? 1 : 0 ;
nt = nt / nc; dd = 0 ;
s2t = 1.0f; nmt = 0 ;
, N);
)
at a = nt - nc; b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) *
Tr) R = (D * nnt - N * (dd
E * diffuse;
= true;
efl + refr)) && (depth < MAXDEPTH)
, N);
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbability(diffuse, I,
estimation - doing it properly, closely fol
df;
radiance = SampleLight(&rand, I, &L, &R,
e.x + radiance.y + radiance.z) > 0) &&
w = true;
at brdfPdf = EvaluateDiffuse(L, N);
at3 factor = diffuse * INVPI;
at weight = Mis2(directPdf, brdfPdf
at cosThetaOut = dot(N, L);
E * ((weight * cosThetaOut) / direct
andom walk - done properly, closely foll
ive)
at3 brdf = SampleDiffuse(diffuse, N, r1, r2, &R, &pdf
survive;
pdf;
n = E * brdf * (dot(N, R) / pdf);
sion = true;

Introduction

A Brief History of Many Cores

Once upon a time...

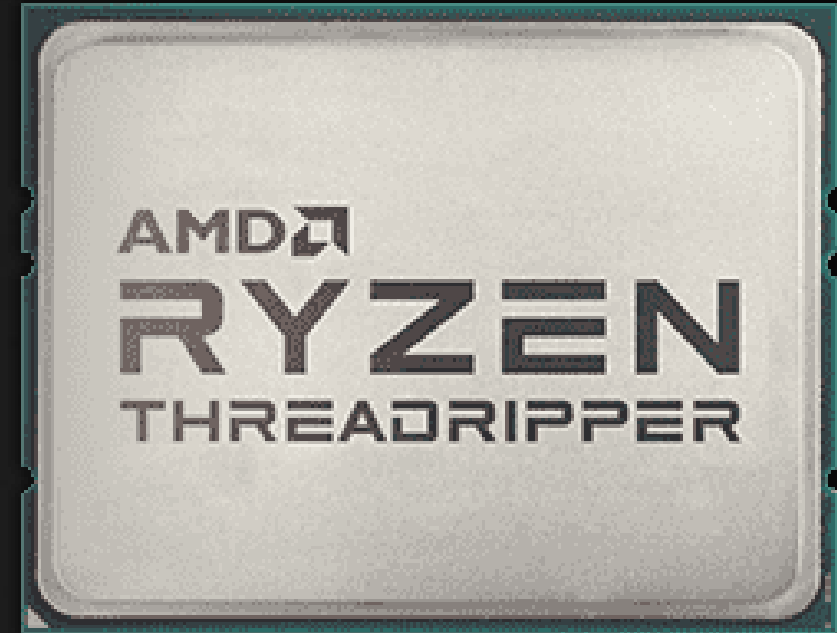
Then, in 2005: Intel’s Core 2 Duo (*April 22*).
 (Also 2005: AMD Athlon 64 X2. *April 21*.)

2007: Intel Core 2 Quad

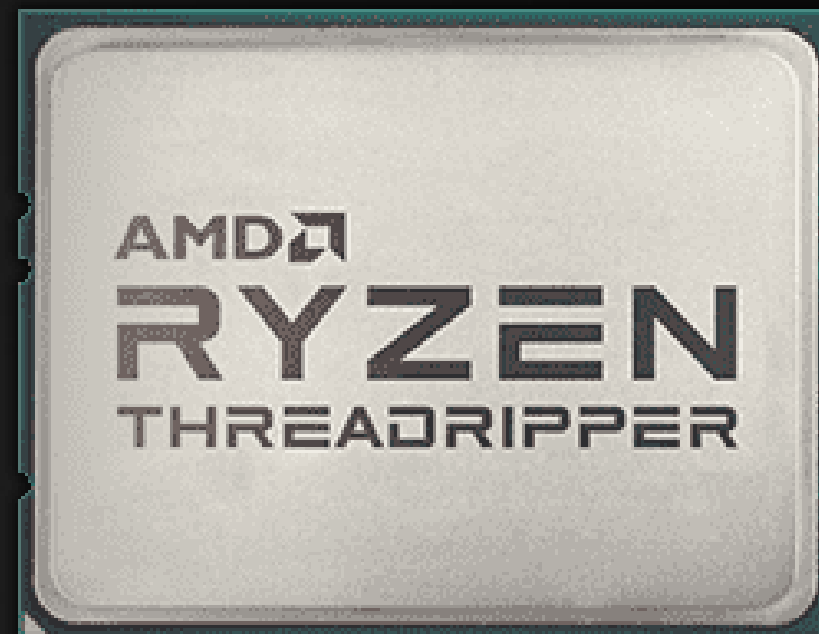
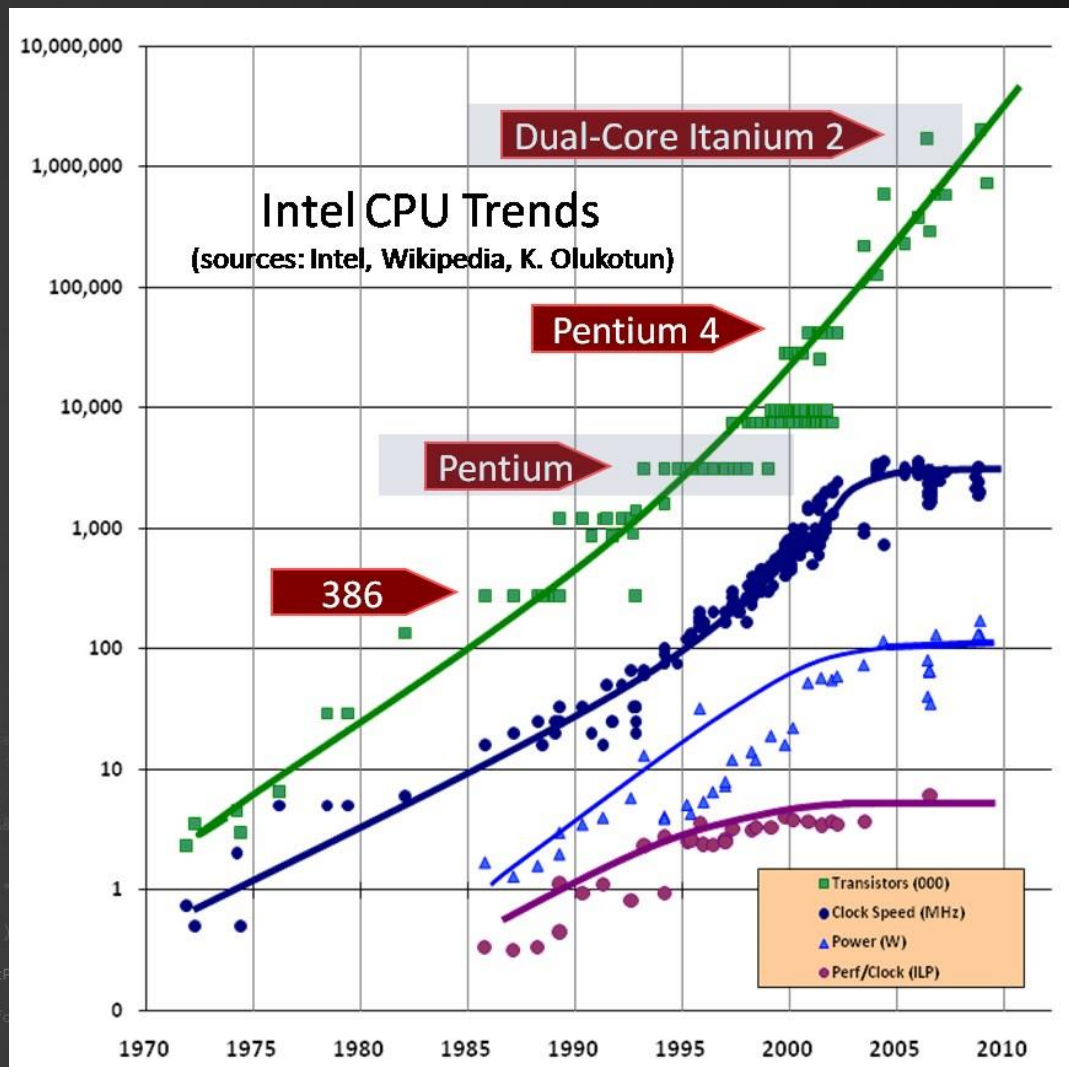
2010: AMD Phenom II X6

2017: Threadripper 1920X

2018: Threadripper 2950X



Introduction



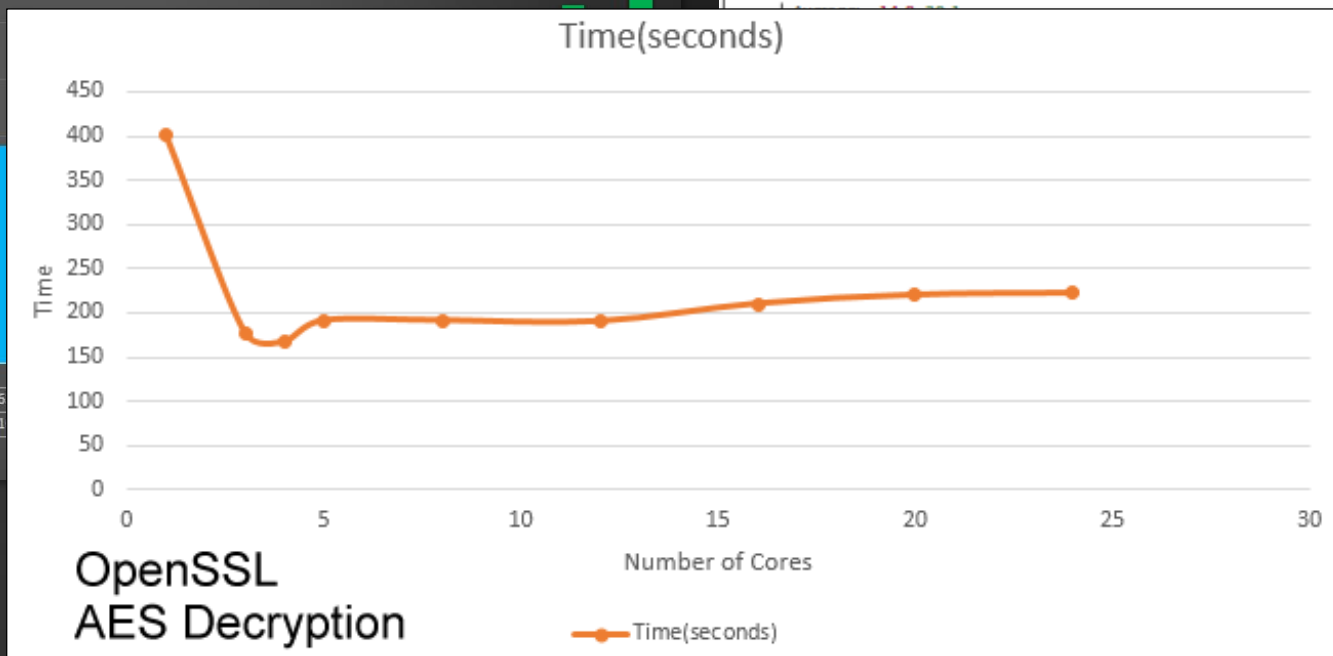
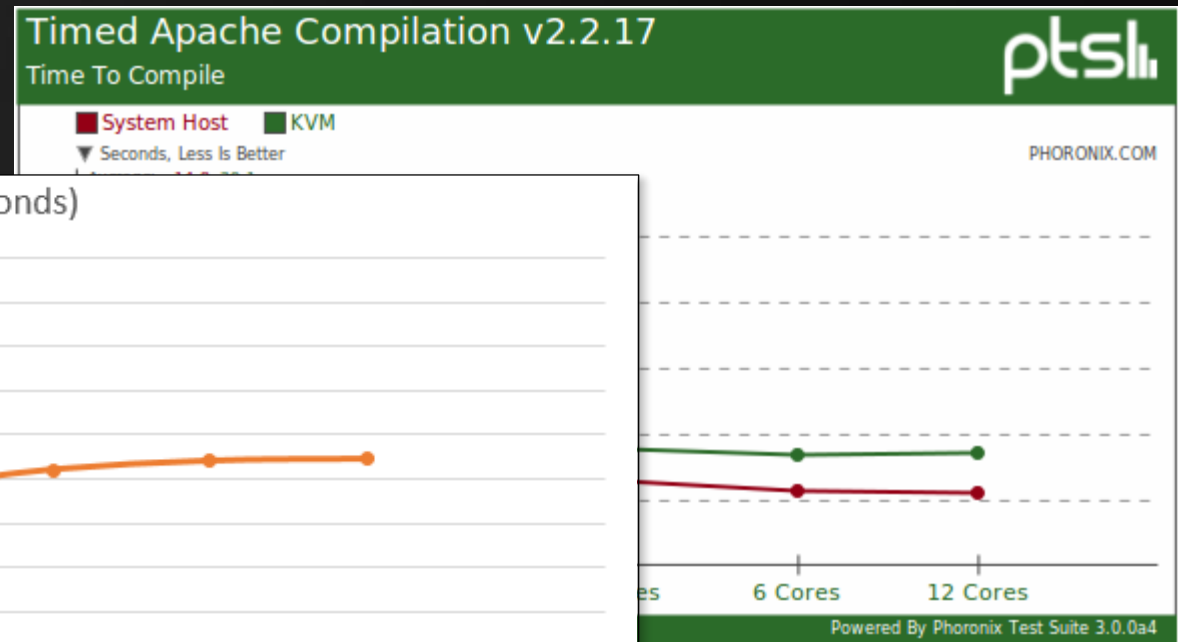
Introduction

Threads / Scalability

```

...
    & (depth < MAXDEPTH)
...
    = insi
    at = nt
    os2t = 1
    D, N );
    )
...
    at a = m
    at Tr =
    (r) R =
...
    E * diffi
    = true;
...
    refl + re
...
    D, N );
    refl * E
    = true;
...
    MAXDEPTH;
...
    survive :
    estimat
    df;
    -radiance
    .x + ra
...
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
    random walk - done properly, closely following Spherical
    (ive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    
```

Performance scaling in HWBOT Prime (java)



Introduction

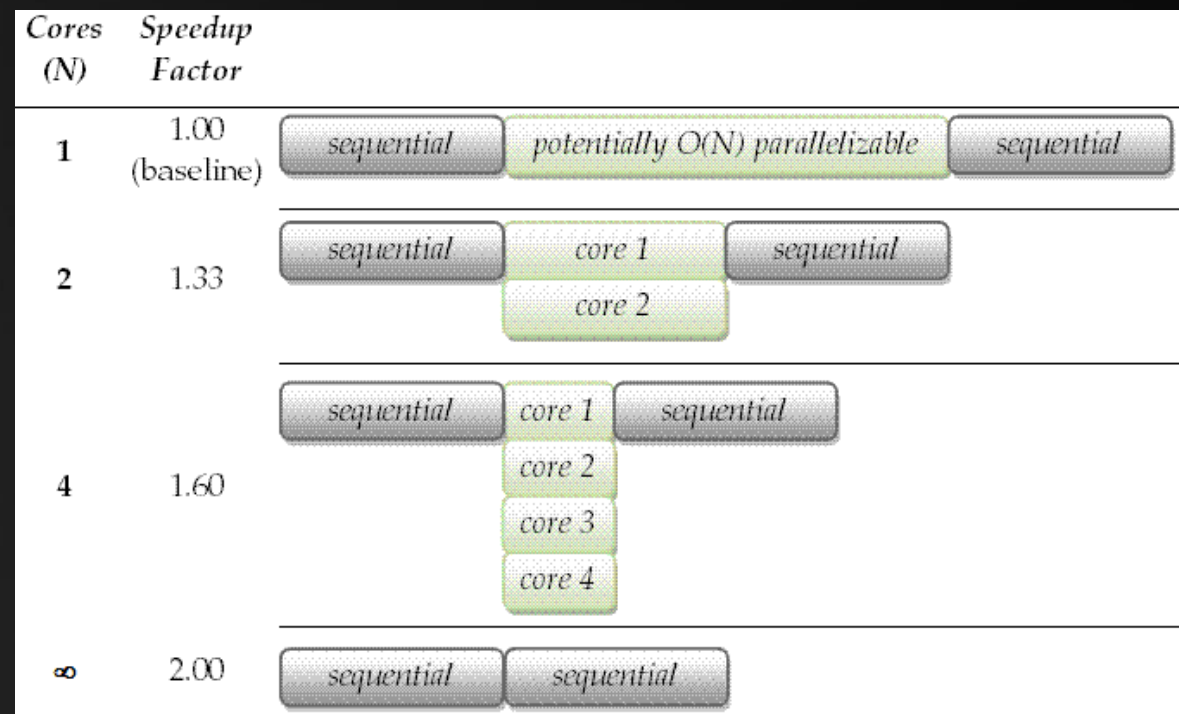
Optimizing for Multiple Cores

What we did before:

1. Profile.
2. Understand the hardware.
3. Trust No One.

Goal:

- ~~■ It's fast enough when it scales linearly with the number of cores.~~
- It's fast enough when the parallelizable code scales linearly with the number of cores.
- It's fast enough if there is no sequential code.



Today's Agenda:

- Introduction
- Hardware
- Trust No One / An Efficient Pattern
- Experiments
- Final Assignment

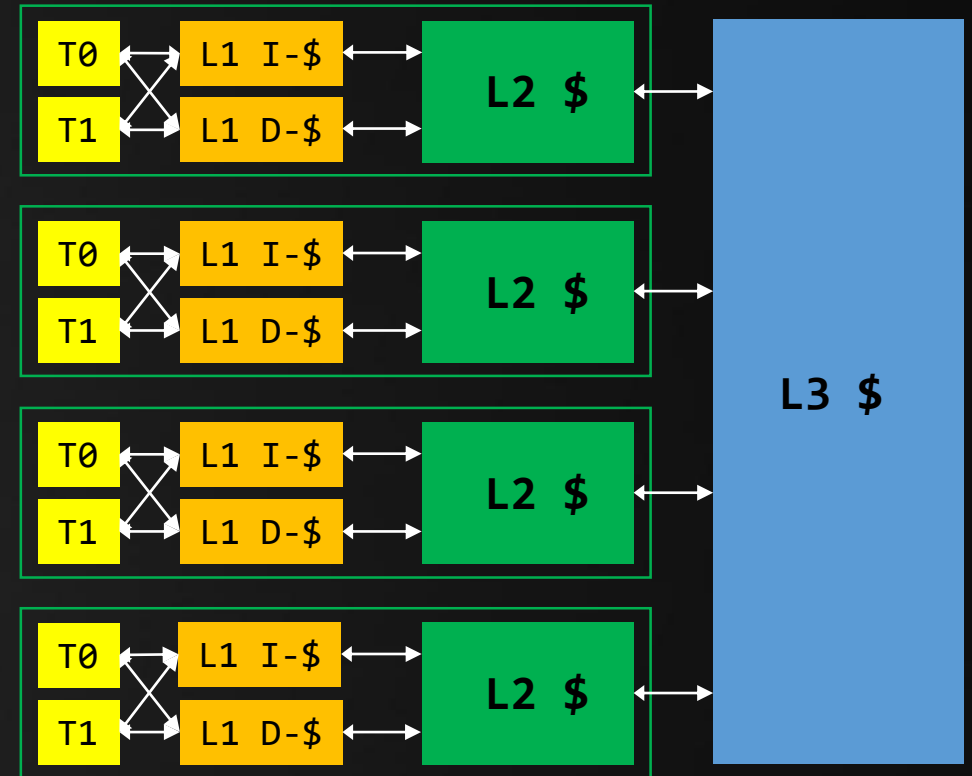


Hardware

Hardware Review

We have:

- Four physical cores
- Each running two threads
- L1 cache: 32Kb, 4 cycles latency
- L2 cache: 256Kb, 10 cycles latency
- A large shared L3 cache.



Hardware

Simultaneous Multi-Threading (SMT)

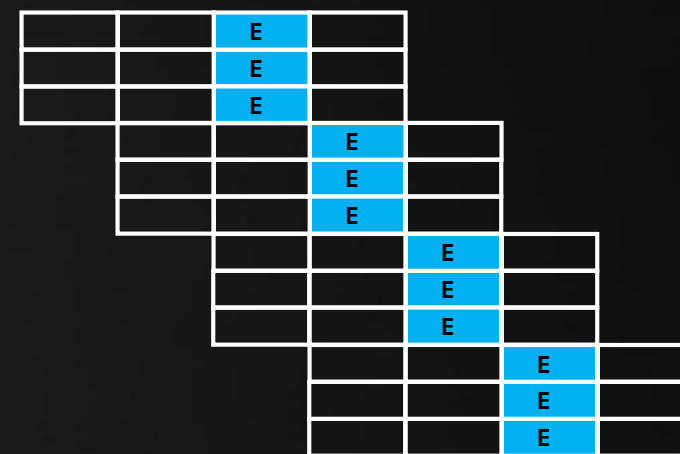
(Also known as hyperthreading)

Pipelines grow wider and deeper:

- Wider, to execute multiple instructions in parallel in a single cycle.
- Deeper, to reduce the complexity of each pipeline stage, which allows for a higher frequency.

However, parallel instructions must be independent, otherwise we get bubbles.

Observation: two independent threads provide twice as many independent instructions.



```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn / nc;
        ps2t = 1.0f - nnt * nnt;
        D, N );
    }
}

at a = nt - nc; b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) *
Tr) R = (D * nnt - N * (ddn
);

E * diffuse;
= true;

efl + refr)) && (depth < MAXDEPTH)
);
refl * E * diffuse;
= true;

MAXDEPTH)
survive = SurvivalProbability( diffuse,
estimation - doing it properly, closely
df;
radiance = SampleLight( &rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && (rand

w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psum
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance

random walk - done properly, closely following 3d
ive)

;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```



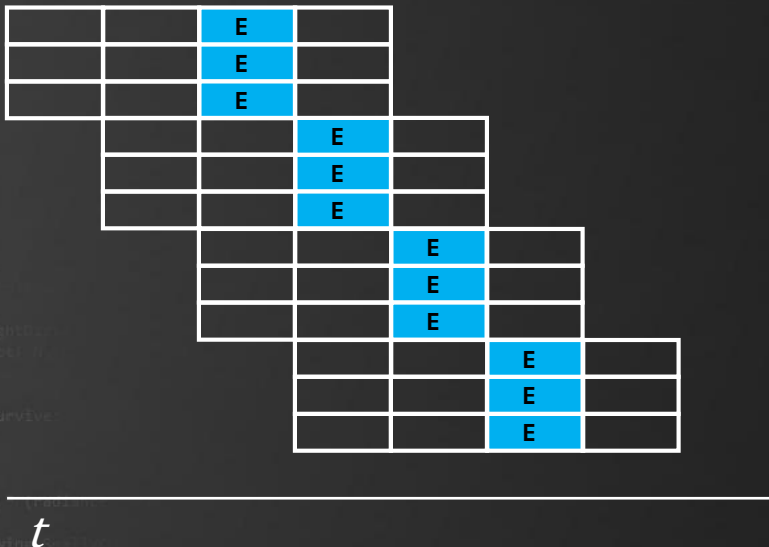
Hardware

Simultaneous Multi-Threading (SMT)

```

...
    &(depth < MAXDEPTH)
...
    inside ? 1 : 0;
    nt = nt / nc; ddn = ddn / nc;
    cos2t = 1.0f - nnt * nnt;
    D, N );
    )
...
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) * R);
    Tr) R = (D * nnt - N * (ddn
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
...
    MAXDEPTH)
    survive = SurvivalProbability( diffuse, I,
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &light
    e.x + radiance.y + radiance.z) > 0) && (cos2t > 0)
...
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf)
...
    random walk - done properly, closely following
    (survive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```



```

fldz
xor ecx, ecx
fld dword ptr [4520h]
mov edx, 28929227h
fld dword ptr [452Ch]
push esi
mov esi, 0C350h
add ecx, edx
mov eax, 91D2A969h
xor edx, 17737352h
shr ecx, 1
mul eax, edx
fld st(1)
faddp st(3), st
mov eax, 91D2A969h
shr edx, 0Eh
add ecx, edx
fmul st(1),st
xor edx, 17737352h
shr ecx, 1
mul eax, edx
shr edx, 0Eh
dec esi
jne tobetimed+1Fh

```



Hardware

Simultaneous Multi-Threading (SMT)

Nehalem (i7): *six* wide.

- Three memory operations
- Three calculations (float, int, vector)

```

ics
& (depth < MAXDEPTH)
    = inside ? 1 : 0;
    nt = nt / nc; ddn = ddn * nc;
    ps2t = 1.0f - nnt * ddn;
    D, N );
    )
    at a = nt - nc, b = nt + nc;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (ddn
    E * diffuse;
    = true;
    -
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
    MAXDEPTH)
    survive = SurvivalProbability( diffuse;
    estimation - doing it propo
    df;
    radiance = SampleLight( &rand
    e.x + radiance.y + radiance.z
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N, &R, Spdf );
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf);
    random walk - done properly, closely following
    (live)
    ;
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, Spdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    
```

execution unit 1 MEM
 execution unit 2 MEM
 execution unit 3 MEM
 execution unit 4 CALC
 execution unit 5 CALC
 execution unit 6 CALC

fld	mov		
mov	mov		
fld			
fldz	add	xor	mul
xor	fld	shr	fmul
push	faddp		



```

fldz
xor ecx, ecx
fld dword ptr [4520h]
mov edx, 28929227h
fld dword ptr [452Ch]
push esi
mov esi, 0C350h
add ecx, edx
mov eax, [91D2h]
xor edx, 17737352h
shr ecx, 1
mul eax, edx
fld st(1)
faddp st(3), st
mov eax, 91D2A969h
shr edx, 0Eh
add ecx, edx
fmul st(1),st
    
```

```

xor edx, 17737352h
shr ecx, 1
mul eax, edx
shr edx, 0Eh
dec esi
jne tobetimed+1Fh
    
```



Hardware

Simultaneous Multi-Threading (SMT)

Nehalem (i7): *six* wide*.

- Three memory operations
- Three calculations (float, int, vector)

SMT: feeding the pipe from *two* threads.

All it really takes is an extra set of registers.

```

...
    &(depth < MAXDEPTH)
...
    inside ? 1 : 0;
    nt = nt / nc;
    ns2t = 1.0f / nnt;
    D, N );
...
    at a = nt - nc, b = nt - nc;
    at Tr = 1 - (R0 + (1 - R0)
    Tr) R = (D * nnt - N * (dd
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
...
    D, N );
    refl * E * diffuse;
    = true;
...
    MAXDEPTH)
    survive = SurvivalProbability( diffuse
    estimation - doing it propo
    df;
    radiance = SampleLight( &rand
    e.x + radiance.y + radiance.z
...
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N,
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf)
...
    random walk - done properly, closely following
    (ive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2);
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;

```

execution unit 1 MEM
 execution unit 2 MEM
 execution unit 3 MEM
 execution unit 4 CALC
 execution unit 5 CALC
 execution unit 6 CALC

fld	mov		
mov	mov		
fld			
fldz	add	xor	mul
xor	fld	shr	fmul
push	faddp		



```

fldz
xor ecx, ecx
fld dword ptr [4520h]
mov edx, 28929227h
fld dword ptr [452Ch]
push esi
mov esi, 0C350h
add ecx, edx
mov eax, [91D2h]
xor edx, 17737352h
shr ecx, 1
mul eax, edx
fld st(1)
faddp st(3), st
mov eax, 91D2A969h
shr edx, 0Eh
add ecx, edx
fmul st(1),st
xor edx, 17737352h
shr ecx, 1
mul eax, edx
shr edx, 0Eh
dec esi
jne tobetimed+1Fh

```

```

fld st(1)
faddp st(3), st
mov eax, 91D2A969h
shr edx, 0Eh
add ecx, edx
fmul st(1),st
xor edx, 17737352h
shr ecx, 1
mul eax, edx
shr edx, 0Eh
dec esi
fldz
xor ecx, ecx
fld dword ptr [4520h]
mov edx, 28929227h
fld dword ptr [452Ch]
push esi
mov esi, 0C350h
add ecx, edx
mov eax, [91D2h]
xor edx, 17737352h
shr ecx, 1
mul eax, edx
jne tobetimed+1Fh

```

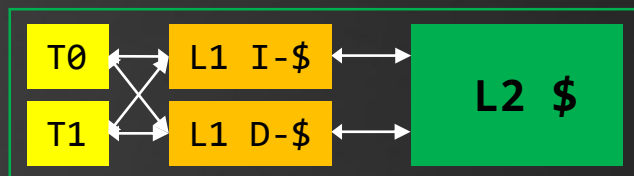
*Details: The Architecture of the Nehalem Processor and Nehalem-EP SMP Platforms, Thomadakis, 2011.



Hardware

Simultaneous Multi-Threading (SMT)

Hyperthreading does mean that now *two* threads are using the same L1 and L2 cache.



- For the average case, this will reduce data locality.
- If both threads use the same data, data locality remains the same.
- One thread can also be used to fetch data that the other thread will need *.

*: Tolerating Memory Latency through Software-Controlled Pre-Execution in Simultaneous Multithreading Processors, Luk, 2001.



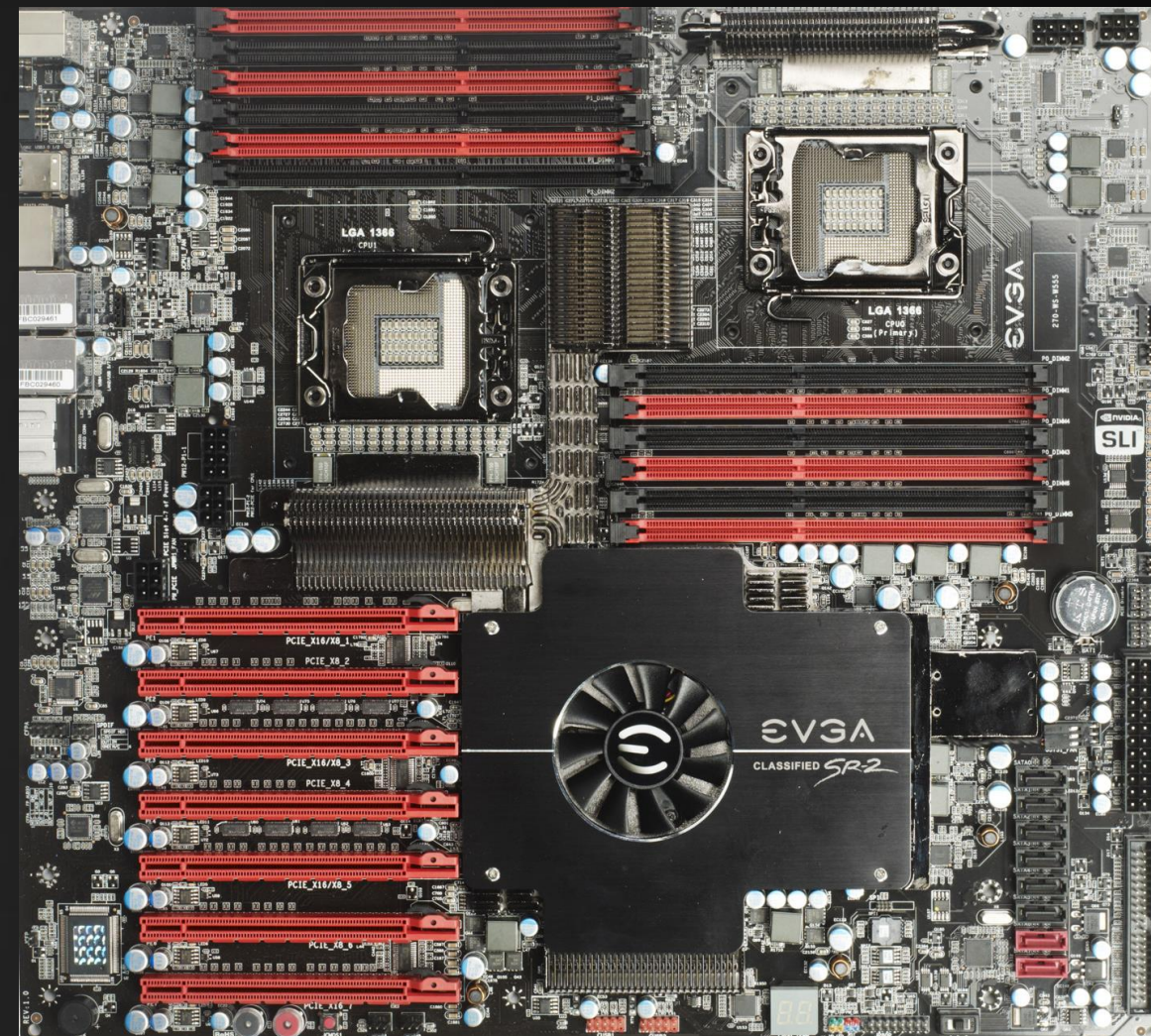
Hardware

Multiple Processors: NUMA

Two physical processors on a single mainboard:

- Each CPU has its own memory
- Each CPU can access the memory of the other CPU.

The penalty for accessing ‘foreign’ memory is ~50%.

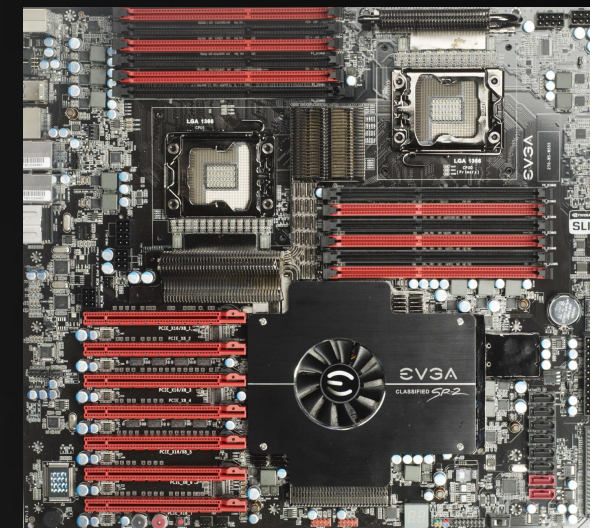
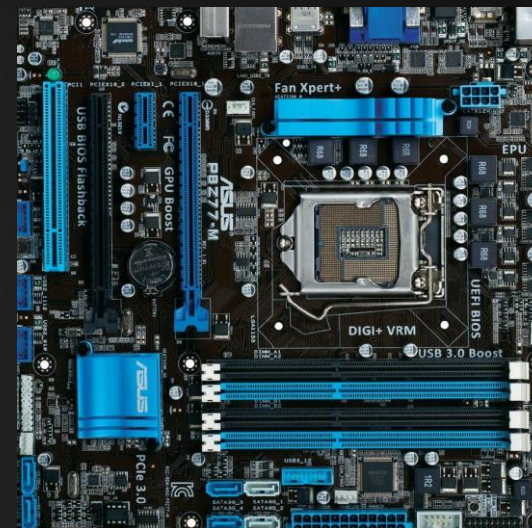


Hardware

Multiple Processors: NUMA

Do we care?

- Most boards host 1 CPU.
- A quadcore still talks to memory via a single interface.

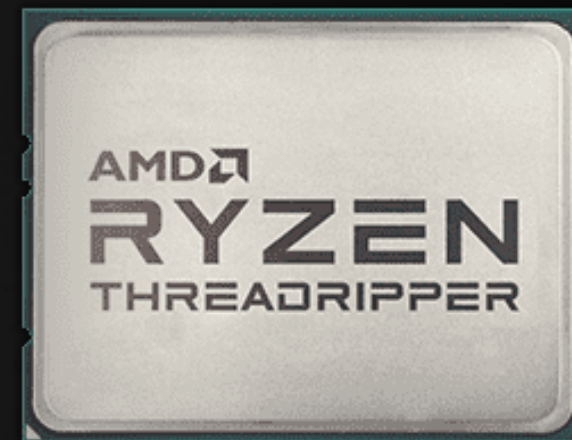


However:

Threadripper is a NUMA device.

Threadripper = 2x Zeppelin, with for each Zeppelin:

- L1, L2, L3 cache
- A link to memory



This CPU behaves as two CPUs in a single socket.

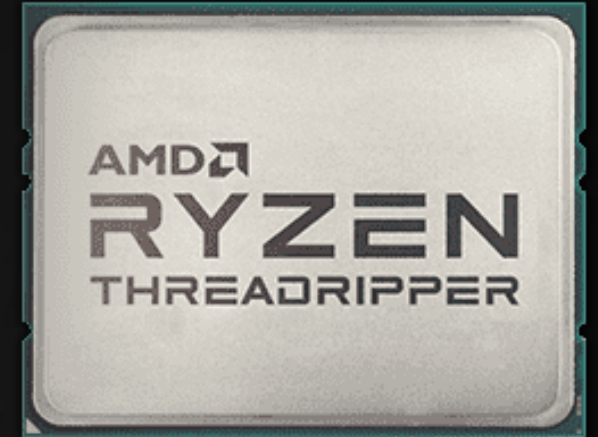


Hardware

Multiple Processors: NUMA

Threadripper & Windows:

- Threadripper hides NUMA from the OS
- Most software is not NUMA-aware.



```

ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * nc;
        ps2t = 1.0f - nnt * nnt;
        D, N );
    }
}

at a = nt - nc; b = nt * nc;
at Tr = 1 - (R0 + (1 - R0) * R);
Tr) R = (D * nnt - N * (ddn * nnt));

E * diffuse;
= true;

-
efl + refr) && (depth < MAXDEPTH)
D, N );
refl * E * diffuse;
= true;

MAXDEPTH)
survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light);
e.x + radiance.y + radiance.z) > 0) && (survive)
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
random walk - done properly, closely following
ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```



Today's Agenda:

- Introduction
- Hardware
- Trust No One / An Efficient Pattern
- Experiments
- Final Assignment



Trust No One

Windows

```

DWORD WINAPI myThread(LPVOID lpParameter)
{
    unsigned int& myCounter = *((unsigned int*)lpParameter);
    while(myCounter < 0xFFFFFFFF) ++myCounter;
    return 0;
}

int main(int argc, char* argv[])
{
    using namespace std;
    unsigned int myCounter = 0;
    DWORD myThreadId;
    HANDLE myHandle = CreateThread(0, 0, myThread, &myCounter;, 0, &myThreadId);
    char myChar = ' ';
    while(myChar != 'q') {
        cout << myCounter << endl;
        myChar = getch();
    }
    CloseHandle(myHandle);
    return 0;
}

```



Trust No One

Boost

```

#include <boost/thread.hpp>
#include <boost/chrono.hpp>
#include <iostream>

void wait(int seconds)
{
    boost::this_thread::sleep_for(boost::chrono::seconds{seconds});
}

void thread()
{
    for (int i = 0; i < 5; ++i)
    {
        wait(1);
        std::cout << i << '\n';
    }
}

int main()
{
    boost::thread t{thread};
    t.join();
}

```



Trust No One

OpenMP

```

#pragma omp parallel for
for( int n = 0; n < 10; ++n ) printf( " %d", n );
printf( ".\n" );

float a[8], b[8];
#pragma omp simd
for( int n = 0; n < 8; ++n) a[n] += b[n];

struct node { node *left, *right; };
extern void process(node* );
void postorder_traverse(node* p)
{
    if (p->left)
        #pragma omp task
        postorder_traverse(p->left);
    if (p->right)
        #pragma omp task
        postorder_traverse(p->right);
    #pragma omp taskwait
    process(p);
}

```



Trust No One

Intel TBB

```
#include "tbb/task_group.h"

using namespace tbb;

int Fib( int n )
{
    if (n<2)
    {
        return n;
    }
    else
    {
        int x, y;
        task_group g;
        g.run( [&]{x=Fib( n - 1 );} ); // spawn a task
        g.run( [&]{y=Fib( n - 2 );} ); // spawn another task
        g.wait(); // wait for both tasks to complete
        return x + y;
    }
}
```



Trust No One

Considerations

When using external tools to manage your threads, ask yourself:

- What is the overhead of creating / destroying a thread?
- Do I even know when threads are created?
- Do I know on which cores threads execute?

What if... we handled everything ourselves?

```

ics
& (depth < MAXDEPTH)
{
    int n = inside ? 1 : 0;
    int nt = nt / nc;
    double ddn = ddn * 0.5;
    double s2t = 1.0f - nnt * ddn;
    double D, N );
}

at a = nt - nc; b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * ddn);
Tr) R = (D * nnt - N * (ddn * nnt + ddn));

E * diffuse;
= true;

efl + refr)) && (depth < MAXDEPTH)
{
    double D, N );
    refl * E * diffuse;
    = true;
}

MAXDEPTH)
{
    survive = SurvivalProbability( diffuse );
    estimation - doing it properly, closely following
    if;
    radiance = SampleLight( &rand, I, &L, &light );
    e.x + radiance.y + radiance.z) > 0) && (depth < MAXDEPTH)
    {
        w = true;
        at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
        at3 factor = diffuse * INVPI;
        at weight = Mis2( directPdf, brdfPdf );
        at cosThetaOut = dot( N, L );
        E * ((weight * cosThetaOut) / directPdf) * (radiance
    }

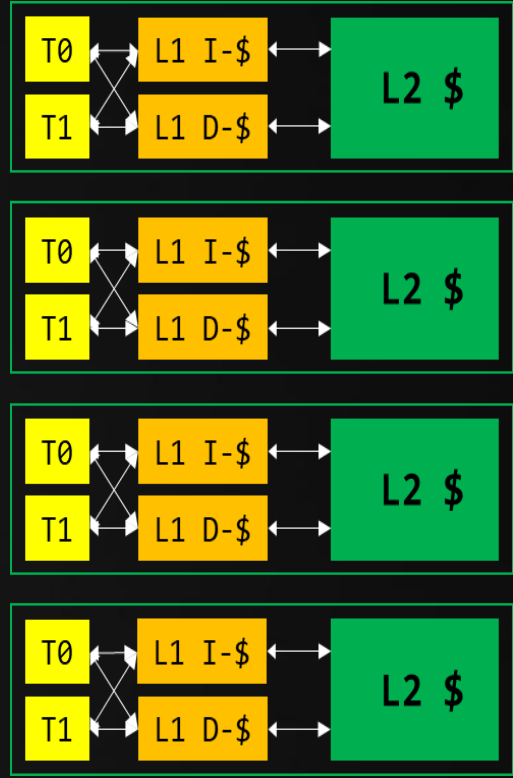
random walk - done properly, closely following
ive)
{
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
}
    
```



Trust No One

```

ics
& (depth < MAXDEPTH)
c = inside ? i
nt = nt / nc; do
os2t = 1.0f - nm
D, N );
0);
at a = nt - nc;
at Tr = 1 - (R0 + (1 - R0)
Tr) R = (D * nnt - N * (D0)
E * diffuse;
= true;
efl + refr)) && (d
D, N );
refl * E * diffuse;
= true;
MAXDEPTH)
survive = SurvivalProbab
estimation - doing
df;
radiance = SampleLight( &rand, I, &L, &light
e.x + radiance.y + radiance.z) > 0) && (rand
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psum
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
random walk - done properly, closely following S
ive)
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
    
```



tasks:



- Worker threads never die
- Tasks are claimed by worker threads
- Execution of a task may depend on completion of other tasks
- Tasks can produce new tasks



Trust No One

```

...
    & (depth < MAXDEPTH)
...
    = inside ? i
    nt = nt / nc; do
    os2t = 1.0f - nm
    D, N );
    );
...
    at a = nt - nc;
    at Tr = 1 - (R0 + (1 - R0)
    Tr) R = (D * nnt - N * (D0)
...
    E * diffuse;
    = true;
...
    efl + refr)) && (d
...
    D, N );
    efl * E * diffuse;
    = true;
...
    MAXDEPTH)
...
    survive = SurvivalProbability(
    estimation - doing
    if;
    radiance = SampleLight( &rand, I, &L, &light
    e.x + radiance.y + radiance.z) > 0) && (cosTheta
...
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psum;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
...
    random walk - done properly, closely following 3D
    vive)
...
    at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf
    survive;
    pdf;
    n = E * brdf * (dot( N, R ) / pdf);
    sion = true;
    
```

worker thread 0

worker thread 1

worker thread 2

worker thread 3

worker thread 4

worker thread 5

worker thread 6

worker thread 7

Fibers:

- Light-weight threads, with a complete state: registers (incl. program counter), stack
- Available in Windows, PS4, ...
- Allows the task system to suspend a job, e.g. to wait for scheduled sub-tasks

Sub-tasks:

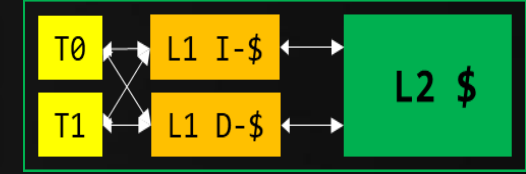
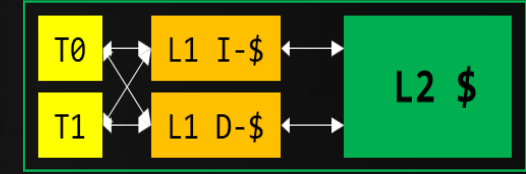
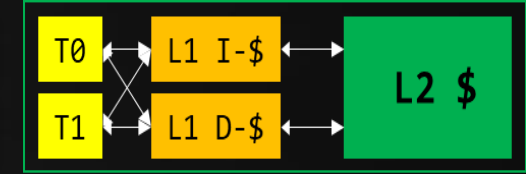
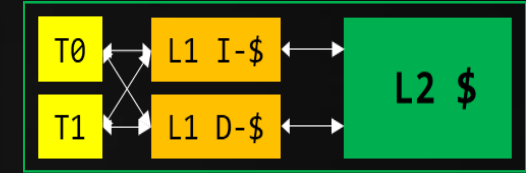
- Decrement a counter when done
- When counter reaches zero, linked task is resumed.

tasks:



Naughty Dog’s “The Last of Us”:

- Tasks are executed as *fibers*
- A fiber stores a stack and a set of registers
- Tasks can be interrupted by storing the fiber in a *waiting list*



Today's Agenda:

- Introduction
- Hardware
- Trust No One / An Efficient Pattern
- Experiments
- Final Assignment



Experiments

Experiments

```

...
    & (depth < MAXDEPTH)
...
    = inside ? 1.0f : 0.0f;
    nt = nt / nc; ddn = ddn * ddn;
    cos2t = 1.0f - nnt * ddn;
    D, N );
    )
...
    at a = nt - nc, b = nt * nc;
    at Tr = 1 - (R0 + (1 - R0) *
    Tr) R = (D * nnt - N * (ddn
...
    E * diffuse;
    = true;
...
    refl + refr)) && (depth < MAXDEPTH)
    D, N );
    refl * E * diffuse;
    = true;
...
MAXDEPTH)
survive = SurvivalProbability( diffuse, 1);
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light);
e.x + radiance.y + radiance.z) > 0) && (depth <
w = true;
at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
at3 factor = diffuse * INVPI;
at weight = Mis2( directPdf, brdfPdf );
at cosThetaOut = dot( N, L );
E * ((weight * cosThetaOut) / directPdf) * (radiance
...
random walk - done properly, closely following Section 2.6.2
vive)
...
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf);
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;

```

1. False sharing

- Setups:
 - 8 threads update a single counter
 - 8 threads update counters in a single cache line
 - 8 threads update counters in different cache lines

2. Locking to cores

- Four rotating hedgehogs, core-locked and not core-locked

3. Calculating the Mandelbrot using worker threads

4. Hyperthreading

- Setup:
 - 4 threads calculate the special Mandelbrot
 - 8 threads calculate the special Mandelbrot
 - Now with worker threads
 - Switch out Mandelbrot for blur, to test bandwidth-intensive app



Today's Agenda:

- Introduction
- Hardware
- Trust No One / An Efficient Pattern
- Experiments
- Final Assignment



/INFOMOV/

END of “Multithreading”

next lecture: “Guest Lecture”

```
ics
& (depth < MAXDEPTH)
{
    if (inside ? 1 : 0)
    {
        nt = nt / nc; ddn = ddn * ddn;
        cos2t = 1.0f - nnt * nnt;
        D, N );
    }
}

at a = nt - nc; b = nt + nc;
at Tr = 1 - (R0 + (1 - R0) * ddn);
Tr) R = (D * nnt - N * (ddn * nnt));

E * diffuse;
= true;

efl + refr) && (depth < MAXDEPTH)
{
    D, N );
    refl * E * diffuse;
    = true;
}

MAXDEPTH)

survive = SurvivalProbability( diffuse );
estimation - doing it properly, closely following
if;
radiance = SampleLight( &rand, I, &L, &light);
e.x + radiance.y + radiance.z) > 0) && (cosThetaOut > 0)
{
    w = true;
    at brdfPdf = EvaluateDiffuse( L, N ) * Psurvive;
    at3 factor = diffuse * INVPI;
    at weight = Mis2( directPdf, brdfPdf );
    at cosThetaOut = dot( N, L );
    E * ((weight * cosThetaOut) / directPdf) * (radiance
}

random walk - done properly, closely following
(ive)
;
at3 brdf = SampleDiffuse( diffuse, N, r1, r2, &R, &pdf );
survive;
pdf;
n = E * brdf * (dot( N, R ) / pdf);
sion = true;
```

