AMD RYZEN™ CPU OPTIMIZATION

Presented by Ken Mitchell & Elliot Kim
Join AMD ISV Game Engineering team members for an introduction to the AMD Ryzen™ CPU followed by advanced optimization topics. Learn about the “Zen” microarchitecture, power management, and CodeXL profiler. Gain insight into code optimization opportunities using hardware performance-monitoring counters. Examples may include assembly and C/C++.

- Ken Mitchell is a Senior Member of Technical Staff in the Radeon Technologies Group/AMD ISV Game Engineering team where he focuses on helping game developers utilize AMD CPU cores efficiently. Previously, he was tasked with automating & analyzing PC applications for performance projections of future AMD products. He studied computer science at the University of Texas at Austin.

- Elliot Kim is a Senior Member of Technical Staff in the Radeon Technologies Group/AMD ISV Game Engineering team where he focuses on helping game developers utilize AMD CPU cores efficiently. Previously, he worked as a game developer at Interactive Magic and has since gained extensive experience in 3D technology and simulations programming. He holds a BS in Electrical Engineering from Northeastern University in Boston.
Introduction
- Microarchitecture
- Power Management
- Profiler

Optimization
- Compiler
- Concurrency
- Shader Compiler
- Prefetch
- Data Cache
Introduction
Microarchitecture

“Zen”
Updated Feb 28, 2017: Generational IPC uplift for the “Zen” architecture vs. “Piledriver” architecture is +52% with an estimated SPECint_base2006 score compiled with GCC 4.6–O2 at a fixed 3.4GHz. Generational IPC uplift for the “Zen” architecture vs. “Excavator” architecture is +64% as measured with Cinebench R15 1T, and also +64% with an estimated SPECint_base2006 score compiled with GCC 4.6–O2, at a fixed 3.4GHz. System configs: AMD reference motherboard(s), AMD Radeon™ R9 290X GPU, 8GB DDR4-2667 (“Zen”)/8GB DDR3-2133 (“Excavator”)/8GB DDR3-1866 (“Piledriver”), Ubuntu Linux 16.x (SPECint_base2006 estimate) and Windows® 10 x64 RS1 (Cinebench R15). SPECint_base2006 estimates: “Zen” vs. “Piledriver” (31.5 vs. 20.7 | +52%), “Zen” vs. “Excavator” (31.5 vs. 19.2 | +64%). Cinebench R15 1T scores: “Zen” vs. “Piledriver” (139 vs. 79 both at 3.4G | +76%), “Zen” vs. “Excavator” (160 vs. 97.5 both at 4.0G | +64%). GD-108
### Relationship Masks:

<table>
<thead>
<tr>
<th>Processor</th>
<th>F</th>
<th>E</th>
<th>D</th>
<th>C</th>
<th>B</th>
<th>A</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core/L1/L1D/L2U</td>
<td>C00</td>
<td>300</td>
<td>C00</td>
<td>300</td>
<td>C0</td>
<td>30</td>
<td>C</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L3U</td>
<td>FF00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Package/NumaNode</td>
<td>FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Use Case Benefits:

- **High Gaming FPS**
  - Consoles have 8 physical cores without SMT – Ryzen™ 7 gives you 8 physical cores with SMT!
- **Fast Digital Content Creation**
- **Fast Compile Times**
## MICROARCHITECTURE

### COMPETITIVE FREQUENCIES FOR HIGH END DESKTOP WITH SIXTEEN LOGICAL PROCESSORS

<table>
<thead>
<tr>
<th>ACPI XPSS</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pstate0</td>
<td>3600</td>
</tr>
<tr>
<td>Pstate1</td>
<td>3200</td>
</tr>
<tr>
<td>Pstate2</td>
<td>2200</td>
</tr>
</tbody>
</table>

### Precision Boost State

<table>
<thead>
<tr>
<th>Description</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>The max frequency the part can run at when 6+ physical cores are in CC6 idle. Subject to core-to-core variance, part-to-part variance, and temperature.</td>
<td>4100</td>
</tr>
<tr>
<td>The expected average frequency of a typical, 1 threaded application.</td>
<td>4000</td>
</tr>
<tr>
<td>The max frequency the part can run at with all cores active.</td>
<td>3700</td>
</tr>
<tr>
<td>The expected average frequency of a typical, fully threaded application.</td>
<td>3600</td>
</tr>
</tbody>
</table>

- High precision tuning with 25MHz increments
- Actual frequency is limited by TDP, current, and temperature
- AMD Ryzen™ 7 1800X shown
## Caches:

<table>
<thead>
<tr>
<th>Level</th>
<th>Count</th>
<th>Capacity</th>
<th>Sets</th>
<th>Ways</th>
<th>Line Size</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>uop</td>
<td>8</td>
<td>2 K uops</td>
<td>32</td>
<td>8</td>
<td>8 uops</td>
<td>NA</td>
</tr>
<tr>
<td>L1I</td>
<td>8</td>
<td>64 KB</td>
<td>256</td>
<td>4</td>
<td>64 B</td>
<td>4 clocks</td>
</tr>
<tr>
<td>L1D</td>
<td>8</td>
<td>32 KB</td>
<td>64</td>
<td>8</td>
<td>64 B</td>
<td>4 clocks</td>
</tr>
<tr>
<td>L2</td>
<td>8</td>
<td>512 KB</td>
<td>1024</td>
<td>8</td>
<td>64 B</td>
<td>17 clocks</td>
</tr>
<tr>
<td>L3U</td>
<td>2</td>
<td>8 MB</td>
<td>8192</td>
<td>16</td>
<td>64 B</td>
<td>40 clocks</td>
</tr>
</tbody>
</table>
Cache Latency (less is better)

- L1D
- L2U
- L3U

- AMD Ryzen
- Intel Skylake
### Microarchitecture

#### Instruction Set Evolution

| Year | Family | Product Family | Architecture | Example Model | ADX | CLFLUSHOPT | RDSEED | SMAP | XGETBV | XSAVES | AES | FMA | F16C | AVX | AVX2 | MOVBE | BMX | RDRND | SMEP | FSGSBASE | XSAVEOPT | BMI | FMA | XSAVEC | XSAVES |
|------|--------|----------------|--------------|---------------|-----|-------------|--------|------|--------|--------|-----|-----|------|-----|------|-------|-----|-------|-------|-----|-----|--------|-------|
| 2017 | 17h    | “Summit Ridge” | “Zen”        | Ryzen 7 1800X | 1   | 1           | 1      | 1    | 1      | 1      | 1   | 1   | 1    | 1   | 1    | 1     | 1   | 1      | 1     | 1   | 1    | 1      | 1   |
| 2015 | 15h    | “Carrizo”/“Bristol Ridge” | “Excavator” | A12-9800 | 0   | 0           | 0      | 0    | 0      | 0      | 1   | 1   | 1    | 1   | 1    | 1     | 1   | 1      | 1     | 1   | 1    | 1      | 1   |
| 2014 | 15h    | “Kaveri”/“Godavari” | “Steamroller” | A10-7890K | 0   | 0           | 0      | 0    | 0      | 0      | 1   | 1   | 1    | 1   | 1    | 1     | 1   | 1      | 1     | 1   | 1    | 1      | 1   |
| 2012 | 15h    | “Vishera” | “Piledriver” | FX-8370 | 0   | 0           | 0      | 0    | 0      | 0      | 0   | 0   | 0    | 0   | 0    | 0     | 0   | 1      | 1     | 1   | 1    | 1      | 1   |
| 2011 | 15h    | “Zambezi” | “Bulldozer” | FX-8150 | 0   | 0           | 0      | 0    | 0      | 0      | 0   | 0   | 0    | 0   | 0    | 0     | 0   | 1      | 1     | 1   | 1    | 1      | 1   |
| 2013 | 16h    | “Kabini” | “Jaguar” | A6-1450 | 0   | 0           | 0      | 0    | 0      | 0      | 0   | 0   | 0    | 0   | 0    | 0     | 0   | 0      | 1     | 1   | 1    | 1      | 1   |
| 2011 | 14h    | “Ontario” | “Bobcat” | E-450 | 0   | 0           | 0      | 0    | 0      | 0      | 0   | 0   | 0    | 0   | 0    | 0     | 0   | 0      | 1     | 0   | 0    | 0      | 0   |
| 2011 | 12h    | “Llano” | “Husky” | A8-3870 | 0   | 0           | 0      | 0    | 0      | 0      | 0   | 0   | 0    | 0   | 0    | 0     | 0   | 0      | 0     | 0   | 0    | 0      | 0   |
| 2009 | 10h    | “Greyhound” | “Greyhound” | Phenom II X4 955 | 0   | 0           | 0      | 0    | 0      | 0      | 0   | 0   | 0    | 0   | 0    | 0     | 0   | 0      | 0     | 0   | 0    | 0      | 0   |

- ADX multi precision support
- CLFLUSHOPT Flush Cache Line Optimized SFENCE order
- RDSEED Pseudorandom number generation Seed
- SHA Secure Hash Algorithm (SHA-1, SHA-256)
- SMAP Supervisor Mode Access Prevention
- XGETBV Get extended control register
- XSAVES Compact and Supervisor Save/Restore

+ CLZero Zero Cache Line

- FMA4
- TBM
- XOP
IO Hub has 24 lanes of PCIe® 3.0 (pending PCIe certification)

- x16 GPU
- x4 AMD Chipset
- x4 Storage

Example clocks:
- CCLK 3.6 GHz
- MemClk 1.3 GHz (DDR4-2667)
- LClk 600 MHz
All structures available in 1T mode

Front End Queues are round robin with priority overrides

high throughput from SMT

+41% performance in Cinebench R15 nT with SMT enabled*

* Based on pre-release Ryzen 7 8C16T running at fixed 3.6GHz. nT Score SMT Off: 1150. nT Score SMT On: 1617. Gain: 40.6%. Test system: AMD Internal Reference motherboard, Radeon™ R9 290X GPU, Windows® 10 x64, Radeon Software 16.12, 8GB DDR4-2667.
MICROARCHITECTURE
PERFORMANCE MONITORING COUNTER DOMAINS

- IC/BP: instruction cache and branch prediction
- DE: instruction decode, dispatch, microcode sequencer, & micro-op cache
- EX (SC): integer ALU & AGU execution and scheduling
- FP: floating point
- LS: load/store
- L2
- L3
- DF: Data Fabric
- UMC: Unified Memory Controller (NDA only)
- IOHC: IO Hub Controller (NDA only)
- rdpmc
- SMN in/out
Power Management
The default power management settings are recommended for normal operation to help achieve maximum active performance and lowest idle power.

- High Performance Power Scheme
  - Maximum processor performance
  - All logical processors are in Pstate0 only
- Core Parking Disabled
  - OS scheduler may use any processor & prefers physical cores
- Balanced Power Scheme
  - Utilization determines processor performance
  - All logical processors may change Pstates
- Core Parking Enabled
  - OS scheduler eligible processor range limited by utilization

Windows® 10 tickless idle improves boost performance

- Profiling tools using small sampling intervals can increase activity which may reduce effective boost frequency.
  - Precision Boost latency
  - ~1 ms to change frequency
- Windows 10 Timeout Intervals
  - Platform Timer Resolution 15.6ms default, 1ms games
  - PerfIncTime & PerfDecTime 1*30ms
  - CPIlluminateTime (to unpark) 3*30ms
  - CPDewilluminateTime (to park) 10*30ms
- Software Profiler Sampling Intervals
  - Microsoft xperf 1ms
  - CodeXL Power Profiler GUI & CLI 10ms
Disabling power management can reduce variation during AB testing.

**BIOS Settings**

- "Zen" Common Options
  - Custom Core Pstates
    - Disable all except Pstate0
    - Set a reasonable frequency & voltage such as P0 custom default
    - Note SMU may still reduce frequency if application exceeds power, current, thermal limits
  - Core Performance Boost = Disable
  - Global C-state Control = Disable

- Use High Performance power scheme
CODEXL

- [https://github.com/GPUOpen-Tools/CodeXL](https://github.com/GPUOpen-Tools/CodeXL)

**CodeXL v2.3 Features**
- CPU Custom
  - Time-based Sampling
  - Instruction-based sampling
    - **All IBS op samples**
    - All IBS fetch samples
  - Events by Hardware Source
    - Core rdpmc events only
- Power Profiler
  - Logical core\d+ Avg Frequency (MHz)
  - RAPL Package Energy (mJ)
  - Core\d+ RAPL Core Energy (mJ)

**Instruction Based Sampling (IBS)** can be more accurate than traditional sampling. Use IBS.
- Traditional sampling, used by Events by Hardware Source, attributes performance data for a window of time to the instruction pointer after the expiration of a user defined sampling interval.
- IBS selects a random instruction fetch or micro-op after the expiration of a user defined sampling interval. When the fetch or micro-op completes, an ISR is called to store the performance data.
- IBS is disabled by default to improve performance.
  - BIOS > Zen Common Options > Enable IBS=Enable

<table>
<thead>
<tr>
<th>Traditional sampling</th>
<th>window0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBS op sampling</td>
<td>uop0</td>
</tr>
</tbody>
</table>
Ryzen™ 7 1800X shown

CODEXL
POWER PROFILER

Ryzen™ 7 1800X shown
Frequency profiling
- `pushd "C:\Program Files (x86)\CodeXL"
- CodeXLPowerProfiler.exe -P frequency -o c:\logs\freq.csv -d 60
- `popd`

Energy profiling
- `pushd "C:\Program Files (x86)\CodeXL"
- CodeXLPowerProfiler.exe -P energy -o c:\logs\energy.csv -d 60
- `popd`

Frequency + Energy profiling
- `pushd "C:\Program Files (x86)\CodeXL"
- CodeXLPowerProfiler.exe -P frequency -P energy -o c:\logs\freqAndEnergy.csv -d 60
- `popd`
<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
<th>IBS all ops</th>
<th>IBS tag-to-ret</th>
<th>IBS comp-to-ret</th>
<th>IBS BR</th>
<th>IBS msp BR</th>
<th>IBS taken BR</th>
<th>IBS msp taken BR</th>
<th>IBS RET</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3D12Multithreading:WorkItemThread(int)</td>
<td>D3D12Multithreading.exe</td>
<td>17,234</td>
<td>35,100</td>
<td>16,676</td>
<td>155</td>
<td>1</td>
<td>155</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Direct3D:XMMatrixLookAtLH(ret, _1, _2, _3)</td>
<td>D3D12Multithreading.exe</td>
<td>23</td>
<td>1,657</td>
<td>1,349</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>FrameResourceWriteConstantBuffers(struct D3D...</td>
<td>D3D12Multithreading.exe</td>
<td>13</td>
<td>952</td>
<td>720</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>D3D12Multithreading:SetCommonPipelinedState...</td>
<td>D3D12Multithreading.exe</td>
<td>10</td>
<td>2,560</td>
<td>1,605</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>D3D12Multithreading:OnRender( void )</td>
<td>D3D12Multithreading.exe</td>
<td>10</td>
<td>1,472</td>
<td>475</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Direct3D:XMMatrixPerspectiveFOV( float,...</td>
<td>D3D12Multithreading.exe</td>
<td>10</td>
<td>539</td>
<td>200</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Camera:GetShadowMapMatrix(struct Direct3DX...</td>
<td>D3D12Multithreading.exe</td>
<td>10</td>
<td>1,472</td>
<td>475</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Functions with a high sample count usually indicate performance bottlenecks. Sort the table according to a specific metric to highlight potential bottleneck functions.
## CODEXL

**ALL IBS FETCH SAMPLES > FUNCTIONS**

![Graphical representation of CODEXL](image)

<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
<th>IBS Fetch</th>
<th>IBS Fetch Killed</th>
<th>IBS Fetch Attempt</th>
<th>IBS Fetch Comp</th>
<th>IBS Fetch Aassert</th>
<th>IBS L1 ITLB Hit</th>
<th>IBS ITLB L</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3D12Multithreading::OnUpdate( void )</td>
<td>D3D12Multithreading.exe</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>DirectX::XMScaleSin CostFloat* float</td>
<td>D3D12Multithreading.exe</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>D3D12Multithreading::SetCommonPipelineState()</td>
<td>D3D12Multithreading.exe</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>FrameResourcesInit( void )</td>
<td>D3D12Multithreading.exe</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>_security_check_cookie</td>
<td>D3D12Multithreading.exe</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>DirectX::XMMatrixLookAtLH( union __m128,union __m128,union __m128)</td>
<td>D3D12Multithreading.exe</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>FrameResources::WriteConstantBuffers()</td>
<td>D3D12Multithreading.exe</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>WinMain</td>
<td>D3D12Multithreading.exe</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D3D12Multithreading::OnRender( void )</td>
<td>D3D12Multithreading.exe</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D3D12Multithreading::OnRender( void )</td>
<td>D3D12Multithreading.exe</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D3D12Applications::ApplicationProc( struct HWND__ )</td>
<td>D3D12Multithreading.exe</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Camera::Camera( void )</td>
<td>D3D12Multithreading.exe</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DirectX::XMMatrixPerspectiveFovRH( float, float, float, float, float, float)</td>
<td>D3D12Multithreading.exe</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x operator new( unsigned int )</td>
<td>D3D12Multithreading.exe</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

*Note: Functions with a high sample count usually indicate performance bottlenecks. Sort the table according to a specific metric to highlight potential bottleneck functions.*
CODEXL
ALL IBS FETCH SAMPLES > SOURCE
Optimization
Many recent AAA games are using old compilers.

But many developers who were using 2012 have already moved to 2015.
<table>
<thead>
<tr>
<th>Year</th>
<th>Visual Studio Changes</th>
<th>AMD Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>Added nullptr keyword. Replaced VCBuild with MSBuild.</td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td>Added x64 native compiler.</td>
<td>“K8”</td>
</tr>
</tbody>
</table>
CONCURRENCY

PROOF BY EXAMPLE THAT GAMES CAN USE 16 LOGICAL PROCESSORS

* Results based on use of the Ashes of the Singularity game’s benchmark.
PROOF BY EXAMPLE THAT USING ALL LOGICAL PROCESSORS ISN’T ALWAYS GOOD


- Default NumContexts = 3
  - OnInit calls _beginthreadex for each
- Default Draws = 1025

Recommend
- useSMT option
  - Set default value based on profiling
  - Some applications can benefit from SMT
  - processors=(useSMT)\?logical:physical;
  - NumContexts = min(processors -1, Draws/300)
WinMain + NumContext=7

WinMain + NumContext=8

WinMain has SMT sharing & thread migration
These penalties outweigh work benefits in this case
Shader Compiling
Avoid compiling shaders during game play if possible
- Shader compiling often has high branch misprediction & memory usage

Else
- Use driver shader cache for D3D12, D3D11, Vulkan
- Compile on many threads

D3D11 AGS Shader Compiler Controls
Prefetch

Avoid software prefetch
Avoid software prefetch. Improve Instruction Cache (IC) & Op Cache (OC) hit rate by using efficient hardware prefetch. Allow additional compiler optimizations.

Profile

- Minimize prefetch instructions dispatched
  - Events by Hardware Source / 04Bh [Prefetch Instructions Dispatched] (LsPrefInstrDisp) Load & Store (& not NTA)

Code

- Try removing prefetch intrinsics:
  - _m_prefetchw
  - _m_prefetch
  - _mm_prefetch (except _MM_HINT_NTA)
#include "stdafx.h"
#include "intrin.h"
#include <numeric>
#include <chrono>

#define LEN 40000
alignas(64) double a[LEN];
alignas(64) double b[LEN];
alignas(64) double c[LEN];

void work() {
    for (size_t i = 0; i < LEN / 4; i++) {
#if 1
    _m_prefetchw(&a[i * 4 + 64]);
    _m_prefetch(&b[i * 4 + 64]);
    _m_prefetch(&c[i * 4 + 64]);
#endif
    a[i * 4] = b[i * 4] * c[i * 4];
    a[i * 4 + 1] = b[i * 4 + 1] * c[i * 4 + 1];
    a[i * 4 + 2] = b[i * 4 + 2] * c[i * 4 + 2];
    a[i * 4 + 3] = b[i * 4 + 3] * c[i * 4 + 3];
    }
}

void main(int argc, char *argv[]) {
    using namespace std::chrono;
    double b0 = (argc > 1) ? std::stod(argv[1], NULL) : 1.0;
    double c0 = (argc > 2) ? std::stod(argv[2], NULL) : 1.0;
    std::fill(b, b + LEN, b0);
    std::fill(c, c + LEN, c0);
    high_resolution_clock::time_point t0 = 
        high_resolution_clock::now();
    volatile double r;
    int hash = 0;
    for (size_t iter = 0; iter < 1000; iter++) {
        work();
        r = a[iter%LEN];
        hash = (hash >> 1) ^ (int)r;
    }
    high_resolution_clock::time_point t1 = 
        high_resolution_clock::now();
    duration<double> time_span = 
        duration_cast<
            duration<double>>((t1 - t0));
    printf("time (milliseconds): %lf\n", 
        1000.0 * time_span.count());
    printf("result: %lf\n", r);
    printf("hash: %i\n", hash);
}
## PREFETCH

### EVENTS BY HARDWARE SOURCE FOR BINARY WITH PREFETCH

![Graphical representation of PREFETCH events]

**201 prefetch inst per 1000 Ret inst**

### Table: Prefetch Events

<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
<th>Ret inst</th>
<th>Ret ups</th>
<th>Ret branch</th>
<th>Ret msg</th>
<th>Prefetch inst</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>prefetch</td>
<td>1,000</td>
<td>4,180</td>
<td>213</td>
<td>14</td>
<td>322</td>
</tr>
</tbody>
</table>

*Functions with a high sample count usually indicate performance bottlenecks. Sort the table according to a specific metric to highlight potential bottleneck functions.*
PREFETCH

EVENTS BY HARDWARE SOURCE FOR BINARY WITHOUT PREFETCH

0 prefetch inst per 1000 Ret inst
With software prefetch instructions.
loop not unrolled.

Without software prefetch instructions.
loop unrolled.
Performance of binary compiled with Microsoft Visual Studio 2015 Update 3
– Tested at 3GHz

Binary compiled without prefetch instructions show higher performance.

<table>
<thead>
<tr>
<th>binary</th>
<th>normalized</th>
<th>avg ms</th>
<th>min</th>
<th>max</th>
<th>stdev</th>
<th>cv</th>
<th>samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>compiled with prefetch</td>
<td>100%</td>
<td>25.9</td>
<td>25.4</td>
<td>26.1</td>
<td>0.2</td>
<td>1%</td>
<td>100</td>
</tr>
<tr>
<td>compiled without prefetch</td>
<td>117%</td>
<td>22.1</td>
<td>21.1</td>
<td>23.7</td>
<td>0.3</td>
<td>1%</td>
<td>100</td>
</tr>
</tbody>
</table>
Data Cache

Use Structure of Arrays
Use Structure of Arrays rather than Arrays of Structures to improve locality and data cache hit rate.

**Profile**
- Minimize Data Cache Misses
  - Instruction-based sampling / All IBS op samples / IBS DC Miss

**Code**
- Use
  ```c
  struct S {
    float xs[LEN];
    ...
  } s;
  ```
- Avoid
  ```c
  struct S {
    float x;
    ...
  } a[LEN];
  ```
#include "stdafx.h"
#include <chrono>
#include <algorithm>

#define LEN 64

struct S {
    float x;
    char16_t str[4096];
} a[LEN];

float stdev_p(S a[], size_t len) {
    float sum = 0.0;
    for (size_t i = 0; i < len; ++i) {
        sum += a[i].x;
    }
    float mean = sum / len;
    float sumSq = 0.0;
    for (size_t i = 0; i < len; ++i) {
        sumSq += (a[i].x - mean)*(a[i].x - mean);
    }
    return sqrt(sumSq / len);
}

void main(int argc, char* argv[]) {
    using namespace std::chrono;
    float v = (argc > 1) ? (float)atof(argv[1]) : 1.0f;
    char16_t ch = (argc > 2) ? (argv[2][0]) : u'a';
    for (size_t i = 0; i < LEN; ++i) {
        a[i].x = v;
        std::fill(std::begin(a[i].str), std::end(a[i].str) - 1, ch);
        v += v;
    }
    high_resolution_clock::time_point t0 = \n        high_resolution_clock::now();
    volatile float r;
    int hash = 0;
    for (size_t iter = 0; iter < 1000000; iter++) {
        r = stdev_p(a, LEN);
        hash = (hash >> 1) ^ int(r);
    }
    high_resolution_clock::time_point t1 = \n        high_resolution_clock::now();
    duration<double> time_span = \n        duration_cast<
            duration<double>>(t1 - t0);
    printf("stdev_p (milliseconds): %lf
", \n        1000.0 * time_span.count());
    printf("stdev_p: %f\n", r);
    printf("hash: %1\n", hash);
}
```c
#include "stdafx.h"
#include <chrono>
#include <algorithm>
#define LEN 64

struct S {
    float xs[LEN];
    char16_t strs[LEN][4096];
} s;

float stdev_p(float a[], size_t len) {
    float sum = 0.0;
    for (size_t i = 0; i < len; ++i) {
        sum += a[i];
    }
    float mean = sum / len;
    float sumSq = 0.0;
    for (size_t i = 0; i < len; ++i) {
        sumSq += (a[i] - mean)*(a[i] - mean);
    }
    return sqrt(sumSq / len);
}

void main(int argc, char* argv[]) {
    using namespace std::chrono;
    float v = (argc > 1) ? (float)atof(argv[1]) : 1.0f;
    char16_t ch = (argc > 2) ? (argv[2][0]) : u'a';
    for (size_t i = 0; i < LEN; ++i) {
        s.xs[i] = v;
        std::fill(std::begin(s.strs[i]), std::end(s.strs[i]) - 1, ch);
        v += v;
    }
    high_resolution_clock::time_point t0 = \
        high_resolution_clock::now();
    volatile float r;
    int hash = 0;
    for (size_t iter = 0; iter < 1000000; iter++) {
        r = stdev_p(s.xs, LEN);
        hash = (hash >> 1) ^ int(r);
    }
    high_resolution_clock::time_point t1 = \
        high_resolution_clock::now();
    duration<double> time_span = \
        duration_cast<
            duration<double>>(t1 - t0);
    printf("stdev_p (milliseconds): %lf\n", \
        1000.0 * time_span.count());
    printf("stdev_p: %f\n", r);
    printf("hash: %1\n", hash);
}
```

DATA CACHE

STRUCTURE OF ARRAYS SOURCE CODE
DATA CACHE

IBS OF ARRAY OF STRUCTURES BINARY

41% DC miss
DATA CACHE
IBS OF STRUCTURE OF ARRAYS BINARY

<table>
<thead>
<tr>
<th>Function</th>
<th>Module</th>
<th>IBS all ops</th>
<th>IBS tag-to-ret</th>
<th>IBS misp BR</th>
<th>IBS misp taken BF</th>
<th>IBS load/store</th>
<th>IBS DC miss</th>
<th>IBS DC hit</th>
<th>IBS misalign acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>stdv_pfillost</td>
<td>aosvssae.exe</td>
<td>1,658</td>
<td>267,528</td>
<td></td>
<td></td>
<td>558</td>
<td>558</td>
<td>558</td>
<td></td>
</tr>
<tr>
<td>main</td>
<td>aosvssae.exe</td>
<td>63</td>
<td>9,756</td>
<td></td>
<td></td>
<td>15</td>
<td>15</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>sqrtf</td>
<td>aosvssae.exe</td>
<td>7</td>
<td>853</td>
<td></td>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Functions with a high sample count usually indicate performance bottlenecks. Sort the table according to a specific metric to highlight potential bottleneck functions.

0% DC miss
DATA CACHE

MSVS2015U3 DISASM

; Arrays of Structures
; stdev_p sum
addss      xmm8,dword ptr [rax+rcx]
addss      xmm8,dword ptr [rax+rcx+2004h]
addss      xmm8,dword ptr [rax+rcx+4008h]
addss      xmm8,dword ptr [rax+rcx+600Ch]
addss      xmm8,dword ptr [rax+rcx+8010h]
addss      xmm8,dword ptr [rax+rcx+0A014h]
addss      xmm8,dword ptr [rax+rcx+0C018h]
addss      xmm8,dword ptr [rax+rcx+0E01Ch]
add         rax,10020h
cmp         rax,80100h
jb          0000000140001040

; Structures of Arrays
; stdev_p sum
addss      xmm8,dword ptr [rax-8]
addss      xmm8,dword ptr [rax-4]
addss      xmm8,dword ptr [rax]
addss      xmm8,dword ptr [rax+4]
addss      xmm8,dword ptr [rax+8]
addss      xmm8,dword ptr [rax+0Ch]
addss      xmm8,dword ptr [rax+10h]
addss      xmm8,dword ptr [rax+14h]
add         rax,20h
sub         rcx,1
jne         0000000140001040

Good locality
stride of 4

Poor locality
stride of 2004h (8196) and +rcx
Performance of binary compiled with Microsoft Visual Studio 2015 Update 3

- Tested at 3GHz

Structure of Arrays binary shows higher performance.

<table>
<thead>
<tr>
<th>binary</th>
<th>normalized</th>
<th>avg ms</th>
<th>min</th>
<th>max</th>
<th>stdev</th>
<th>cv</th>
<th>samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array of Structures</td>
<td>100%</td>
<td>149</td>
<td>149</td>
<td>162</td>
<td>2</td>
<td>1%</td>
<td>100</td>
</tr>
<tr>
<td>Structure of Arrays</td>
<td>135%</td>
<td>111</td>
<td>111</td>
<td>112</td>
<td>0</td>
<td>0%</td>
<td>100</td>
</tr>
</tbody>
</table>
The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION
© 2017 Advanced Micro Devices, Inc. All rights reserved. AMD, Ryzen, and the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Vulkan and the Vulkan logo are registered trademarks of Khronos Group Inc. Microsoft and Windows are registered trademarks of Microsoft Corporation. PCIe is a registered trademark of PCI-SIG. Other names are for informational purposes only and may be trademarks of their respective owners.
Ken Mitchell
- Kenneth.Mitchell@amd.com
- @kenmitchellken

Elliot Kim
- Elliot.Kim@amd.com