Hierarchical Roofline on AMD Instinct™ MI200 GPUs

Noah Wolfe, Xiaomin Lu
October 11, 2022
Agenda

- Introduction
- Roofline Fundamentals
- Empirical Hierarchical Roofline on MI200
  - Roofline Overview
  - Roofline Arithmetic
  - Empirical Roofline Benchmarking
- Roofline Analysis Workflow
  - Tooling
  - Features
  - Bottleneck Analysis Recipe
- Examples
  - Add/Mul/FMA
  - N-Body
- HPC Application Results
### AMD Fueling the Era of Exascale

**OAK RIDGE FRONTIER**

**LAWRENCE LIVERMORE EL CAPITAN**

### AMD INSTINCT™ MI250X ACCELERATOR

- TSMC 6NM TECHNOLOGY
- UP TO 110 CU PER GRAPHICS COMPUTE DIE
- 4 MATRIX CORES PER COMPUTE UNIT
- MATRIX CORES ENHANCED FOR HPC
- 8 INFINITY FABRIC LINKS PER DIE
- SPECIAL FP32 OPS FOR DOUBLE THROUGHPUT

### FRONTIER NODE AT A GLANCE

- Optimized 3rd Gen AMD EPYC™ processor
- Four Instinct MI250X accelerators
- Coherent connectivity
  - Via Infinity Fabric™ interconnect
  - Tightly integrated
  - Unified memory space
Roofline – All Workloads

Orange: Synthetic Workload  Yellow: Proxy app  Green: Full app

Performance (GFLOPS/sec)

Add  Mul  Triad  FMA

Arithmetic Intensity (FLOP/Byte)

countD (HACC)  cmD (HACC)  scatt. (Kripke)  sweep (Kripke)  L* (Kripke)  Pop. (Kripke)  Mul  Triad  FMA

XSBench (OpenMC)  Mini-HACC-256 (HACC)  Mini-HACC-f2 (HACC)  Nbody-FP64  FMA-30  FMA-20  FMA-60  FMA-1024

*MI250x. RESULTS MAY VARY. SEE ENDNOTE: MI200-62
Background – What is Roofline
Background – What is Roofline

- Attainable FLOPs/s
  - FLOPs/s rate as measured empirically on a given device
  - FLOP = floating point operation
  - FLOP counts for common operations
    - Add: 1 FLOP
    - Mul: 1 FLOP
    - FMA: 2 FLOP
  - FLOPs/s = Number of floating-point operations performed per second
Background – What is Roofline

- **Arithmetic Intensity (AI)**
  - characteristic of the workload indicating how much compute (FLOPs) is performed per unit of data movement (Byte)
  - Ex: \( x[i] = y[i] + c \)
    - FLOPs = 1
    - Bytes = \( 1 \times RD + 1 \times WR = 4 + 4 = 8 \)
    - \( AI = \frac{1}{8} \)

![Diagram of Arithmetic Intensity (FLOPs/Byte) vs. Attainable FLOPs/s]

Attainable FLOPs/s

Arithmetic Intensity (FLOPs/Byte)
Background – What is Roofline

- Log-Log plot
  - makes it easy to doodle, extrapolate performance along Moore’s Law, etc...
Background – What is Roofline

- **Roofline Limiters**
  - **Compute**
    - Peak FLOPs/s
  - **Memory BW**
    - AI * Peak GB/s

- **Note:**
  - These are empirically measured values
  - Different SKUs will have unique plots
  - Individual devices within a SKU will have slightly different plots based on thermal solution, system power, etc.
  - MIBench uses suite of simple kernels to empirically derive these values
  - These are **NOT** theoretical values indicating peak performance under “unicorn” conditions
Background – What is Roofline

- Attainable FLOPs/s = 
  \[ \min \left\{ \frac{\text{Peak FLOPs/s}}{AI \times \text{Peak GB/s}} \right\} \]

- Machine Balance:
  - Where \( AI = \frac{\text{Peak FLOPs/s}}{\text{Peak GB/s}} \)
  - Typical machine balance: 5-10 FLOPs/B
    - 40-80 FLOPs per double to exploit compute capability
  - MI250x machine balance: ~16 FLOPs/B
    - 128 FLOPs per double to exploit compute capability
Background – What is Roofline

- Attainable FLOPs/s = \(
\min \left\{ \frac{\text{Peak FLOPs/s}}{\text{AI}} \cdot \frac{\text{Peak GB/s}}{} \right\}
\)

- Machine Balance:
  - Where \( AI = \frac{\text{Peak FLOPs/s}}{\text{Peak GB/s}} \)

- Five Performance Regions:
  - Unattainable Compute
Background – What is Roofline

- Attainable FLOPs/s = \[
\min \left\{ \frac{\text{Peak FLOPs/s}}{\text{AI} \times \text{Peak GB/s}} \right\}
\]

- Machine Balance:
  - Where \( AI = \frac{\text{Peak FLOPs/s}}{\text{Peak GB/s}} \)

- Five Performance Regions:
  - Unattainable Compute
  - Unattainable Bandwidth
Background – What is Roofline

- Attainable FLOPs/s = $\min \left\{ \frac{\text{Peak FLOPs/s}}{\text{AI} \times \text{Peak GB/s}} \right\}$

- Machine Balance:
  - Where $\text{AI} = \frac{\text{Peak FLOPs/s}}{\text{Peak GB/s}}$

- Five Performance Regions:
  - Unattainable Compute
  - Unattainable Bandwidth
  - Compute Bound

![Diagram of Roofline Analysis]

- Unattainable performance (greater than peak FLOPs/s)
- Compute Bound
- HBM GB/s
- Unattainable performance (insufficient bandwidth)
- Peak FLOPs/s

Arithmetic Intensity (FLOPs/Byte) vs. Attainable FLOPs/s graph
Background – What is Roofline

- Attainable FLOPs/s =
  - \( \min \left\{ \frac{\text{Peak FLOPs/s}}{\text{AI} \times \text{Peak GB/s}} \right\} \)

- Machine Balance:
  - Where \( \text{AI} = \frac{\text{Peak FLOPs/s}}{\text{Peak GB/s}} \)

- Five Performance Regions:
  - Unattainable Compute
  - Unattainable Bandwidth
  - Compute Bound
  - Bandwidth Bound
### Background – What is Roofline

- **Attainable FLOPs/s** =
  \[ \min \left\{ \frac{\text{Peak FLOPs/s}}{AI \times \text{Peak GB/s}} \right\} \]

- **Machine Balance:**
  - Where \( AI = \frac{\text{Peak FLOPs/s}}{\text{Peak GB/s}} \)

- **Five Performance Regions:**
  - Unattainable Compute
  - Unattainable Bandwidth
  - Compute Bound
  - Bandwidth Bound
  - Poor Performance
Background – What is Roofline

- Attainable FLOPs/s =
  \[ \text{min} \left\{ \frac{\text{Peak FLOPs/s}}{\text{AI} \times \text{Peak GB/s}} \right\} \]

- Final result is a single roofline plot presenting the peak attainable performance (in terms of FLOPs/s) on a given device based on the arithmetic intensity of any potential workload.

- We have an application independent way of measuring and comparing performance on any platform.
Background – What is “Good” Performance

- Example:
  - We run a number of kernels and measure FLOPs/s
Background – What is “Good” Performance

- Example:
  - We run a number of kernels and measure FLOPs/s
  - Sort kernels by arithmetic intensity
Example:
- We run a number of kernels and measure FLOPs/s
- Sort kernels by arithmetic intensity
- Compare performance relative to hardware capabilities
Example:
- We run a number of kernels and measure FLOPs/s
- Sort kernels by arithmetic intensity
- Compare performance relative to hardware capabilities
- Kernels near the roofline are making good use of computational resources
  - Kernels can have low performance (FLOPS/s), but make good use of BW

Arithmetic Intensity (FLOPs/Byte)

Attainable FLOPs/s

Peak FLOPs/s

HBM GB/s

Compute Bound
Background – What is “Good” Performance

- Example:
  - We run a number of kernels and measure FLOPs/s
  - Sort kernels by arithmetic intensity
  - Compare performance relative to hardware capabilities
  - Kernels near the roofline are making good use of computational resources
    - Kernels can have low performance (FLOPS/s), but make good use of BW
  - Increase arithmetic intensity when bandwidth limited
    - Reducing data movement increases AI
  - Kernels not near the roofline should* have optimizations that can be made to get closer to the roofline
Empirical Hierarchical Roofline on MI200 - Overview

- Peak MFMA GFLOP/sec
- Peak VALU GFLOP/sec
- Peak HBM BW
- Peak L2 BW
- Peak LDS BW
- Peak vL1D BW
- Workload Perf: (GFLOP/sec, AI)
Empirical Roofline Benchmarking (MIPerf)

- Empirical Roofline Benchmarking
  - Measure achievable Peak FLOPS
    - VALU: F32, F64
    - MFMA: F16, BF16, F32, F64
  - Measure achievable Peak BW
    - LDS
    - Vector L1D Cache
    - L2 Cache
    - HBM

- Internally developed micro benchmark algorithms
  - Peak VALU FLOP: axpy
  - Peak MFMA FLOP: Matrix multiplication based on MFMA intrinsic
  - Peak LDS/L1D/L2 BW: Pointer chasing
  - Peak HBM BW: Streaming copy
Roofline Arithmetic: Perfmon Counters

- **Weight**
  - ADD: 1
  - MUL: 1
  - FMA: 2
  - Transcendental: 1

- **FLOP Count**
  - VALU: derived from VALU math instructions (assuming 64 active threads)
  - MFMA: count FLOP directly, in unit of 512

- **Transcendental Instructions (7 in total)**
  - $e^x$, $\log(x)$: F16, F32
  - $\frac{1}{x}$, $\frac{1}{\sqrt{x}}$: F16, F32, F64
  - $\sin x$, $\cos x$: F16, F32

- **Profiling Overhead**
  - Require 3 application replays

### Weight
- ADD: 1
- MUL: 1
- FMA: 2
- Transcendental: 1

### FLOP Count
- VALU: derived from VALU math instructions (assuming 64 active threads)
- MFMA: count FLOP directly, in unit of 512

### Transcendental Instructions (7 in total)
- $e^x$, $\log(x)$: F16, F32
- $\frac{1}{x}$, $\frac{1}{\sqrt{x}}$: F16, F32, F64
- $\sin x$, $\cos x$: F16, F32

### Profiling Overhead
- Require 3 application replays

---

<table>
<thead>
<tr>
<th>ID</th>
<th>HW Counter</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SQ_INSTS_VALU_ADD_F16</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>2</td>
<td>SQ_INSTS_VALU_MUL_F16</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>3</td>
<td>SQ_INSTS_VALU_FMA_F16</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>4</td>
<td>SQ_INSTS_VALU_TRANS_F16</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>5</td>
<td>SQ_INSTS_VALU_ADD_F32</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>6</td>
<td>SQ_INSTS_VALU_MUL_F32</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>7</td>
<td>SQ_INSTS_VALU_FMA_F32</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>8</td>
<td>SQ_INSTS_VALU_TRANS_F32</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>9</td>
<td>SQ_INSTS_VALU_ADD_F64</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>10</td>
<td>SQ_INSTS_VALU_MUL_F64</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>11</td>
<td>SQ_INSTS_VALU_FMA_F64</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>12</td>
<td>SQ_INSTS_VALU_TRANS_F64</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>13</td>
<td>SQ_INSTS_VALU_INT32</td>
<td>IOP counter</td>
</tr>
<tr>
<td>14</td>
<td>SQ_INSTS_VALU_INT64</td>
<td>IOP counter</td>
</tr>
<tr>
<td>15</td>
<td>SQ_INSTS_VALU_MFMA_MOPS_I8</td>
<td>IOP counter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>HW Counter</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>SQ_INSTS_VALU_MFMA_MOPS_F16</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>17</td>
<td>SQ_INSTS_VALU_MFMA_MOPS_BF16</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>18</td>
<td>SQ_INSTS_VALU_MFMA_MOPS_F32</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>19</td>
<td>SQ_INSTS_VALU_MFMA_MOPS_F64</td>
<td>FLOP counter</td>
</tr>
<tr>
<td>20</td>
<td>SQ_LDS_IDX_ACTIVE</td>
<td>LDS Bandwidth</td>
</tr>
<tr>
<td>21</td>
<td>SQ_LDS_BANK_CONFLICT</td>
<td>LDS Bandwidth</td>
</tr>
<tr>
<td>22</td>
<td>TCP_TOTAL_CACHE_ACCESSSES_sum</td>
<td>L1D Bandwidth</td>
</tr>
<tr>
<td>23</td>
<td>TCP_TCC_WRITE_REQ_sum</td>
<td>L2 Bandwidth</td>
</tr>
<tr>
<td>24</td>
<td>TCP_TCC_ATOMIC_WITH_RET_REQ_sum</td>
<td>L2 Bandwidth</td>
</tr>
<tr>
<td>25</td>
<td>TCP_TCC_ATOMIC_WITHOUT_RET_REQ_sum</td>
<td>L2 Bandwidth</td>
</tr>
<tr>
<td>26</td>
<td>TCP_TCC_READ_REQ_sum</td>
<td>L2 Bandwidth</td>
</tr>
<tr>
<td>27</td>
<td>TCC_EA_RDREQ_sum</td>
<td>HBM Bandwidth</td>
</tr>
<tr>
<td>28</td>
<td>TCC_EA_RDREQ_32B_sum</td>
<td>HBM Bandwidth</td>
</tr>
<tr>
<td>29</td>
<td>TCC_EA_WRREQ_sum</td>
<td>HBM Bandwidth</td>
</tr>
<tr>
<td>30</td>
<td>TCC_EA_WRREQ_64B_sum</td>
<td>HBM Bandwidth</td>
</tr>
</tbody>
</table>
Roofline Arithmetic: Metrics

\[
\text{Total FLOP} = 64 \times (\text{SQ\_INSTS\_VALU\_ADD\_F16} + \text{SQ\_INSTS\_VALU\_MUL\_F16} + \text{SQ\_INSTS\_VALU\_TRANS\_F16} + 2 \times \text{SQ\_INSTS\_VALU\_FMA\_F16}) \\
+ 64 \times (\text{SQ\_INSTS\_VALU\_ADD\_F32} + \text{SQ\_INSTS\_VALU\_MUL\_F32} + \text{SQ\_INSTS\_VALU\_TRANS\_F32} + 2 \times \text{SQ\_INSTS\_VALU\_FMA\_F32}) \\
+ 64 \times (\text{SQ\_INSTS\_VALU\_ADD\_F64} + \text{SQ\_INSTS\_VALU\_MUL\_F64} + \text{SQ\_INSTS\_VALU\_TRANS\_F64} + 2 \times \text{SQ\_INSTS\_VALU\_FMA\_F64}) \\
+ 512 \times \text{SQ\_INSTS\_VALU\_MFMA\_MOPS\_F16} \\
+ 512 \times \text{SQ\_INSTS\_VALU\_MFMA\_MOPS\_BF16} \\
+ 512 \times \text{SQ\_INSTS\_VALU\_MFMA\_MOPS\_F32} \\
+ 512 \times \text{SQ\_INSTS\_VALU\_MFMA\_MOPS\_F64}
\]

\[
\text{Total IOP} = 64 \times (\text{SQ\_INSTS\_VALU\_INT32} + \text{SQ\_INSTS\_VALU\_INT64})
\]

\[
\text{LDS}_\text{BW} = 32 \times 4 \times (\text{SQ\_LDS\_IDX\_ACTIVE} - \text{SQ\_LDS\_BANK\_CONFLICT})
\]

\[
\text{vL1D}_\text{BW} = 64 \times \text{TCP\_TOTAL\_CACHE\_ACCESSES\_sum}
\]

\[
\text{L2}_\text{BW} = 64 \times \text{TCP\_TCC\_READ\_REQ\_sum} \\
+ 64 \times \text{TCP\_TCC\_WRITE\_REQ\_sum} \\
+ 64 \times (\text{TCP\_TCC\_ATOMIC\_WITH\_RET\_REQ\_sum} + \text{TCP\_TCC\_ATOMIC\_WITHOUT\_RET\_REQ\_sum})
\]

\[
\text{HBM}_\text{BW} = 32 \times (\text{TCC\_EA\_RDREQ\_32B\_sum} + 64 \times (\text{TCC\_EA\_RDREQ\_sum} - \text{TCC\_EA\_RDREQ\_32B\_sum})) \\
+ 32 \times (\text{TCC\_EA\_WRREQ\_sum} - \text{TCC\_EA\_WRREQ\_64B\_sum}) + 64 \times \text{TCC\_EA\_WRREQ\_64B\_sum}
\]

*All calculations are subject to change*
Roofline Analysis: Manual Rocprof

- For those who like getting their hands dirty
- Generate input file
  - See example roof-counters.txt ➔
- Run rocprof
  
  ```bash
  foo@bar:$ rocprof -i roof-counters.txt --timestamp on ./myCoolApp
  ```
- Analyze results
  - Load `results.csv` output file in csv viewer of choice
  - Derive final metric values using equations on previous slide
- Profiling Overhead
  - Requires one application replay for each pmc line
Roofline Analysis Workflow – Tooling (MIPerf)
Roofline Analysis Workflow – Tooling

Workload Profiling

```bash
miperf profile -n mixbench-hip -c "./mixbench-hip-ro"
```

GUI Import

```bash
miperf import -i -n mixbenchhip --path workloads/mixbench-hip/mi200/ -H pavil1 -u amd -t asw
```

CLI Roofline Generation

```bash
miperf analyze --gen-roofline -p workloads/mixbench-hip/mi200/ -c "./mixbench-hip-ro"
```

* All CLI options are subject to change due to fast prototyping. Refer to MI Performance Profiler (MIPerf) - Audacious Software Team - Confluence (amd.com) for the up-to-date documentation.
Roofline Analysis Workflow – Features

- **GUI Analyzer**
  - Histogram
  - Kernel statistics
  - Kernel filtering
  - Baseline comparison
  - Memory hierarchy selection

- **CLI Analyzer**
  - Kernel filtering
  - Memory hierarchy selection
  - x/y-axis range selection
  - Pdf/jpg file export
Roofline analysis is part of the integrated performance analysis on GPUs
- Except for simple Compute and Memory BW bound, in-depth profiling and tracing analysis is needed

- **Compute Bound workloads**
  - **Compute Unit Analysis** (Instruction Mix, pipeline perf)

- **Memory BW Bound workloads**
  - **L2 Cache analysis** (BW, Util, cache hit)
  - **Vector L1D Cache analysis** (BW, Util, cache hit)
  - **LDS analysis** (BW, Util, latency, bank conflict)

- **Memory Latency Bound workloads**
  - **Wavefront Life analysis**
    - Dependency wait, instr issue wait
  - **Vector L1D Cache analysis**
    - vL1D stall/util/bw/latency
  - **Compute Unit analysis**
    - VMEM/SMEM/LDS latency

- **Dispatch Bound workloads**
  - **SPI analysis**
    - Waveslot/LDS/VGPR/SGPR limit
Roofline Examples on AMD Instinct™ MI250X GPU
Roofline Plot – AMD Instinct™ MI250X Accelerators

- **Device:** Instinct MI250X
  - ORNL Frontier GPU
- Instinct MI250X (Dual GCDs)
  - Figure shows single GCD
- Methodology applies to all AMD Instinct MI200 series GPUs

Roofline Example #1 – Add

- **Calculation:**
  - \( a[i] = a[i] + b[i] \)

- **VALU Ops Per Thread:**
  - 1x V_ADD

- **HBM MEM Ops Per Thread:**
  - 2x RD
  - 1x WR

- **Arithmetic Intensity:**
  - 1 FLOP / (3 * 4Byte) = 1/12
Roofline Example #1 – Add

- **Calculation:**
  - \( a[i] = a[i] + b[i] \)

- **Reading two floats for every add results in low arithmetic intensity and HBM limited**
Roofline Example #2 – Mul

- **Calculation:**
  - \( a[i] = x \times b[i] \)

- **VALU Ops Per Thread:**
  - 1x V_MUL

- **HBM MEM Ops Per Thread:**
  - 1x RD
  - 1x WR

- **Arithmetic Intensity:**
  - 1 FLOP / (2 * 4Byte) = 1/8

```
1 template<typename T>
2 __global__ void mul_benchmark(T *buf1, T *buf2, uint32_t nSize)
3 {
4     const uint32_t gid = hipBlockDim_x * hipBlockIdx_x + hipThreadIdx_x;
5     const uint32_t nThreads = gridDim.x * blockDim.x;
6
7     T *a, *b;
8     a = &buf1[gid];
9     b = &buf2[gid];
10    const T x = (T)1.2;
11
12    for(uint32_t offset=0; offset < nSize; offset += nThreads)
13    {
14        a[offset] = x * b[offset];
15    }
```
Roofline Example #2 – Mul

- Calculation:
  - \(a[i] = c \times b[i]\)

- Reading one less float (compared to Add) increases our arithmetic intensity and reduces sensitivity to HBM
Roofline Example #3 – Triad

- **Calculation:**
  - \( a[i] = b[i] + x * a[i] \)

- **VALU Ops Per Thread:**
  - 1x V_ADD
  - 1x V_MUL
  - 1x V_FMA

- **HBM MEM Ops Per Thread:**
  - 2x RD
  - 1x WR

- **Arithmetic Intensity:**
  - \( \frac{2 \text{ FLOP}}{(3 \times 4 \text{Byte})} = \frac{1}{6} \)

```cpp
1 template<typename T>
2 __global__ void triad_benchmark(T *buf1, T *buf2, uint32_t nSize)
3 {
4   const uint32_t gid = hipBlockDim_x * hipBlockIdx_x + hipThreadIdx_x;
5   const uint32_t nThreads = gridDim.x * blockDim.x;
6
7   T *a, *b;
8   a = &buf1[gid];
9   b = &buf2[gid];
10  const T x = (T)1.2;
11
12  for(uint32_t offset=0; offset < nSize; offset += nThreads)
13  {
14    a[offset] = b[offset] + x * a[offset];
15  }
16 }
```
Roofline Example #3 – Triad

- **Calculation:**
  - \( a[i] = b[i] + x \times a[i] \)

- **Performing an extra operation increases arithmetic intensity and further reduces sensitivity to HBM as compared to Add and Mul**
Roofline Example #4 – FMA

- Calculation:
  - \( x = a[i] \times x + y \)

- VALU Ops Per Thread:
  - 1x V_ADD
  - 1x V_MUL

- HBM MEM Ops Per Thread:
  - 1x RD

- Arithmetic Intensity:
  - 2 FLOP / (1 * 4Byte) = 1/2
Roofline Example #4 – FMA

- **Calculation:**
  - $x = a[i] \times x + y$

- Each thread having to load one less value from HBM further increases arithmetic intensity and improves FLOPs/s performance.

*Mi250x. RESULTS MAY VARY. SEE ENDNOTE:** MI200-57, MI200-58, MI200-59, MI200-60
Roofline Example #5 – FMA 1024

- Calculation:
  - \( x = a[i] \times x + y \)

- VALU Ops Per Thread:
  - 1x \text{V\_ADD}
  - 1x \text{V\_MUL}

- HBM MEM Ops Per Thread:
  - 1x \text{RD}

- Arithmetic Intensity:
  - \( 1024 \times 2 \text{ FLOP} / (1 \times 4\text{Byte}) = 512 \)
Roofline Example #5 – FMA 1024

- **Calculation:**
  - $x = a[i] \times x + y$

- Each thread looping over many FMAs with only one read significantly increases arithmetic intensity and becomes compute VALU limited.
**Calculation:**
- \( x = a[i] \times x + y \)

**Further sweeping the number of FMA instructions from 20 to 60 shows the workload transitioning from HBM limited to VALU limited.**
Roofline Use Case - HPC Particle Codes

- Particle interactions form the foundation of many computational science codes from multiple domains
  - **Domains**: Cosmology, astrophysics, molecular dynamics, and more
  - **Applications**: HACC, LAMMPS, NAMD, Amber, GROMACS

**Nbody**

- One such computational algorithm for computing particle interactions leveraged by these applications
- Direct particle-particle method
- Highly accurate
- Computationally expensive ($N^2$)
Roofline Example #1 – Nbody

- Repo: https://github.com/ROCm-Developer-Tools/HIP-Examples/tree/master/mini-nbody/hip
  - Fundamental particle-particle algorithm
  - Single collection of N particles calculating N^2 pair-wise interactions
  - Double precision (FP64)
  - Multiple implementations leveraging different optimization approaches

- “orig”
  - Numerical Computing 101 unoptimized implementation

- “soa”
  - Converting particle data layout from array of structures to structure of arrays

- “block”
  - Loading and computing particle data in “tiles” to increase cache hits

- “unroll”
  - Adding #pragma unroll to particle “tile” processing for loop
Roofline Example #1 – Nbody

- “orig”
  - Numerical Computing 101 unoptimized implementation

- $O(n^2)$ Interaction Ops:
  - 3x V_ADD
  - 6x V_FMA
  - 2x V_MUL
  - 1x V_DIV
  - 1x V_SQRT
  - 3x RD

- $O(n)$ Accumulation Ops:
  - 3x V_FMA
  - 3x RD
  - 3x WR

- Interaction AI:
  - $[(3 + 12 + 2 + 1) \text{FLOPs} / 24 \text{Bytes}] \cdot n^2 = (3/4)n^2$

- Accumulation AI:
  - $(5 \text{ FLOPs} / 24 \text{ Bytes}) \cdot n = n/4$
Roofline Example #1 – Nbody

- "orig"
  - Numerical Computing 101 unoptimized implementation

- Nbody has a very high arithmetic intensity and therefore closer to the top of the roofline (compute sensitive)
- Transcendentals like RSQ do not complete at same rate as ADD, MUL and FMA and therefore limit the peak FLOPS/s performance

*MI250x. RESULTS MAY VARY. SEE ENDNOTE: MI200-61
Roofline Example #1 – Nbody

- “block”
  - Preload a “tile” size worth of particle data into faster shared memory for computing $O(n^2)$ forces

- Processing in “tiles” improves reuse and increases cache hits

```c
typedef struct { double4 *pos, *vel; } BodySystem;

__global__
void bodyForce(double4 *p, double4 *v, double dt, int n) {
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < n) {
        double Fx = 0.0f; double Fy = 0.0f; double Fz = 0.0f;
        for (int tile = 0; tile < gridDim.x; tile++) {
            __shared__ double3 spos[BLOCK_SIZE];
            double4 tpos = p[tile * blockDim.x + threadIdx.x];
            spos[threadIdx.x] = make_double3(tpos.x, tpos.y, tpos.z);
            __syncthreads();

            for (int j = 0; j < BLOCK_SIZE; j++) {
                double dx = spos[j].x - p[i].x;
                double dy = spos[j].y - p[i].y;
                double dz = spos[j].z - p[i].z;
                double distSqr = dx*dx + dy*dy + dz*dz + SOFTENING;
                double invDist = 1.0f / sqrtf(distSqr);
                double invDist3 = invDist * invDist * invDist;

                Fx += dx * invDist3; Fy += dy * invDist3; Fz += dz * invDist3;
            }
            __syncthreads();
        }
        v[i].x += dt*Fx; v[i].y += dt*Fy; v[i].z += dt*Fz;
    }
}
```
Roofline Example #1 – Nbody

- “block”
  - Loading and computing particle data in “tiles” to increase cache hits
  - Working on smaller “tiles” of particles improves cache hits, removing loads from HBM and increasing FLOPs performance
Roofline – All Workloads

**Orange**: Synthetic Workload  **Yellow**: Proxy app  **Green**: Full app

---

- **Add**
- **Mul**
- **Triad**

---

- **XSBench (OpenMC)**
- **Pop. (Kripke)**
- **cmD (HACC)**

---

- **FMA-30**
- **FMA-20**
- **mini-HACC-256 (HACC)**
- **rsqrtf-200**

---

- **L+* (Kripke)**
- **sweep (Kripke)**
- **countD (HACC)**

---

- **min-HACC-f2 (HACC)**

---

**Performance (GFLOPS/sec)** vs **Arithmetic Intensity (FLOP/Byte)**

---

*MI250x: RESULTS MAY VARY. SEE ENDNOTE: MI200-62*
DISCLAIMERS AND ATTRIBUTIONS

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

THIS INFORMATION IS PROVIDED 'AS IS.” 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 NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, 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.

© 2022 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo, Radeon, Instinct, EYPC, Infinity Fabric, ROCm, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.
Math Kernel Endnotes

MI200-57 - Testing Conducted by AMD performance lab as of 4/19/2022 on a single socket optimized 3rd Gen AMD EPYC™ CPU powered server with 1x AMD Instinct™ MI250X OAM (128 GB HBM2e) 560W GPU with AMD Infinity Fabric™ technology resulted in a median score of 92.6 GFLOPS/s on Add Kernel. Server manufacturers may vary configurations, yielding different results. Performance may vary based on use of latest drivers and optimizations.

MI200-58 - Testing Conducted by AMD performance lab as of 4/19/2022 on a single socket optimized 3rd Gen AMD EPYC™ CPU powered server with 1x AMD Instinct™ MI250X OAM (128 GB HBM2e) 560W GPU with AMD Infinity Fabric™ technology resulted in a median score of 149.8 GFLOPS/s on Mul Kernel. Server manufacturers may vary configurations, yielding different results. Performance may vary based on use of latest drivers and optimizations.

MI200-59 - Testing Conducted by AMD performance lab as of 4/19/2022 on a single socket optimized 3rd Gen AMD EPYC™ CPU powered server with 1x AMD Instinct™ MI250X OAM (128 GB HBM2e) 560W GPU with AMD Infinity Fabric™ technology resulted in a median score of 184.7 GFLOPS/s on Triad Kernel. Server manufacturers may vary configurations, yielding different results. Performance may vary based on use of latest drivers and optimizations.

MI200-60 - Testing Conducted by AMD performance lab as of 4/19/2022 on a single socket optimized 3rd Gen AMD EPYC™ CPU powered server with 1x AMD Instinct™ MI250X OAM (128 GB HBM2e) 560W GPU with AMD Infinity Fabric™ technology resulted in a median score of up to 21.7 TFLOPS/s on FMA Kernel. Server manufacturers may vary configurations, yielding different results. Performance may vary based on use of latest drivers and optimizations.
Nbody Endnotes

MI200-61 - Testing Conducted by AMD performance lab as of 4/19/2022 on a single socket optimized 3rd Gen AMD EPYC™ CPU powered server with 1x AMD Instinct™ MI250X OAM (128 GB HBM2e) 560W GPU with AMD Infinity Fabric™ technology resulted in a median score of 8.7 TFLOPS/s on benchmark mini-nbody-orig. Information on mini-nbody-orig: https://github.com/ROCm-Developer-Tools/HIP-Examples/blob/master/mini-nbody/hip/nbody-orig.cpp. Server manufacturers may vary configurations, yielding different results. Performance may vary based on use of latest drivers and optimizations MI200-61
