Part 1 : Roofline Model

Instructor: Leopold Grinberg
IBM, T.J. Watson Research Center, USA

e-mail: leopoldgrinberg@us.ibm.com
The Roofline Model
The Roofline Model

- The roofline model was introduced in 2009 by Williams et.al.

- It provides an easy way to get performance bounds for compute and memory bandwidth bound computations.

- It relies on the concept of Computational Intensity (CI) – sometimes also called Arithmetic or Operational Intensity.

- The Roofline Model provides a relatively simple way for performance estimates based on the computational kernel and hardware characteristics.

  Performance [GF/s] = function (hardware and software characteristics)
FLOPS:Bytes ratio is the basic variable of the Roofline model

\[
\text{for (i=0; i < N; i=i+1)} \quad a[i] = b[i]
\]

\[
\text{for (i=0; i < N; i=i+1)} \quad a[i] = b[i]*b[i]+b[i]
\]

\[
\text{for (i=0; i < N; i=i+1)} \quad a[i] = b[i]*b[i]+\sin(b[i])+\exp(b[i])
\]

DATA TRANSFER, NO FLOPS

DATA TRANSFER, ADDs and MULs

DATA TRANSFER, FLOPS
Performance can be estimated from hardware and kernel characteristics.

Kernels can be Compute bounded (DGEMM) or Communication bounded (DAXPY) (kernels are rarely well balanced).

Some hardware is more communication oriented than another (high memory BW).

Some hardware is more computation oriented than another (high FLOPs).

Mapping kernel characteristics to hardware characteristics (or vice-versa) → performance.
Performance Limiting Factors

DATA

MEMORY BANDWIDTH (READ)

CALCULATIONS (+, -, /, *, ....)

MEMORY BANDWIDTH (WRITE)

FLOPS

DATA
The Roofline Model - is a tool to understand the kernel/hardware limitation and it is also a tool for kernel optimization

**Performance is upper bounded by:**

1) **the peak flop rate**

2) **the streaming bandwidth**
for (i=0; i < N; i=i+1)
a[i] = 2.3*b[i]

for (i=0; i < N; i=i+1)
a[i] = b[i]*b[i]+b[i]

for (i=0; i < N; i=i+1)
a[i] = b[i]*b[i]+sin(b[i])+exp(b[i])

The Roofline Model

Performance [GF/s]

Arithmetic Intensity (FLOPS/BYTE)
FLOPS / Bytes ratio – one of the basic characteristics of a kernel

for (i = 0; i < N; ++i)
    z[i] = x[i] + y[i]

1 ADD
2 (8 byte) loads
1 (8 byte) write
AI = 1 / (2*8 + 8) = 1/24*

for (i = 0; i < N; ++i)
    z[i] = x[i] + y[i]*x[i]

1 ADD
1 MUL
2 (8 byte) loads
1 (8 byte) write
AI = 2 / (2*8 + 8) = 1/12*

for (i = 0; i < N; ++i){
    l1 = A_offset[i]; l2 = A_offset[i+1];
    sum = 0.0
    for (j = 0; j < (l2-l1); ++j)
        sum += A[l1+j] * x[col_index[l2+j]];
    y[i] = sum;
}

1 ADD
1 MUL
2 (8 byte) + 1 (4 bytes) loads
1 (8 byte) write
AI = 2 / (2*8 + 4 + 8) = 1/14

* because of write-allocate traffic on cache-based systems kernel would actually requires an extra read for Z and have even lower AI.
The Roofline Model: Arithmetic Intensity (AI)

Arithmetic Intensity

- BLAS L1, SpMv, stencil
- FFT
- BLAS L3
- Particle methods
The trend is for architectures to have ever decreasing machine balance (the point where the bandwidth roof meets the ceiling moves to the right).

More and more algorithms are going to find themselves memory bound.

Even DGEMM can run into trouble depending on the blocking factor chosen.

A “balanced” architecture can also be a “crippled” one, e.g. low-end GPUs with 1/24th the DP peak performance.

You can achieve a higher percentage of a lower peak.
It is an art to find a perfect match between kernel and hardware characteristics.

In another words it requires a lot of work to create a kernel that will exhaust both, the memory BW and FLOPs capacity at the same time. (many times it is even impossible ....)
Performance depends on how well a given kernel fits node/processor architecture, and/or how well a given kernel is translated by a compiler.

Recall: hardware-kernel characteristics mapping.
Performance depends on how well a given kernel fits node/processor architecture, and/or how well a given kernel is translated by a compiler.

Recall: hardware-kernel characteristics mapping.
N – is large, i.e., buffer does not fit cache

\[
\text{for (i=0; i < N; ++i)} \\
\text{a[i] = buffer[i] + b[i];}
\]

\[
\text{for (i=0; i < N; ++i)} \\
\text{c[i] = buffer[i] + d[i];}
\]

\[\text{Al\_total} = \frac{2}{(2 \times 3 \times 8)} = \frac{1}{24};\]

\[
\text{for (i=0; i < N; ++i)} \\
\text{a[i] = buffer[i] + b[i];} \\
\text{c[i] = buffer[i] + d[i];}
\]

\[\text{Al} = \frac{2}{(5 \times 8)} = \frac{1}{20};\]
sum = 0;
for (i=0; i < N; ++i)
    sum = sum + a[i];

sum0 = sum1 = sum2 = sum3 = 0;
for (i=0; i < N; i+=4){
    sum0 = sum0 + a[i    ];
    sum1 = sum1 + a[i+1];
    sum2 = sum2 + a[i+2];
    sum3 = sum3 + a[i+3];
}
sum0 = sum0+sum1;
sum2 = sum2+sum3;
sum = sum0+sum2;
EXAMPLES and EXERCISES
Example 1: DAXPY

Consider DAXPY: for (i = 0; i < N; ++i) \( y[i] = a\times[i]+y[i] \)

For each “i”: 1 addition, 1 multiplication
2 loads of 8 bytes each
1 store

Execution on BlueGene/Q (Peak 204.8 GFLOP/node)

Performance estimates:

\[ \text{AI} = \frac{2}{3\times8} = \frac{1}{12} \]

\[ \frac{1}{12} < 7.11 \rightarrow \]

We are in the memory BW limited area on the Roofline plot

\[ \frac{7.11}{\frac{1}{12}} = 85.32 \]

\[ \frac{204.8}{85.32} = 2.4 \text{ GF/s} \]
Example 1: DAXPY

Consider DAXPY: for (i = 0; i < N; ++i) \( y[i] = a \times x[i] + y[i] \)

For each “i”:
- 1 addition, 1 multiplication
- 2 loads of 8 bytes each
- 1 store

Execution on BlueGene/Q (Peak 204.8 GFLOP/node):

<table>
<thead>
<tr>
<th># threads</th>
<th>Time [s]</th>
<th>GFLOPS</th>
<th>DDR traffic per node (Bytes/cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0879111</td>
<td>0.455</td>
<td>3.519</td>
</tr>
<tr>
<td>2</td>
<td>0.044039</td>
<td>0.907</td>
<td>7.022</td>
</tr>
<tr>
<td>4</td>
<td>0.022151</td>
<td>1.801</td>
<td>13.94</td>
</tr>
<tr>
<td>8</td>
<td>0.0174019</td>
<td>2.284</td>
<td>17.686</td>
</tr>
<tr>
<td>16</td>
<td>0.017447</td>
<td>2.287</td>
<td>17.719</td>
</tr>
</tbody>
</table>

Performance estimates:

\[ AI = \frac{2}{3 \times 8} = \frac{1}{12} \]

\( \frac{1}{12} < 7 \rightarrow \) We are in the memory BW limited area on the roofline plot

\[ \frac{7.11}{\frac{1}{12}} = 85.32 \]

\[ \frac{204.8}{85.32} = 2.4 \text{ GF/s} \]
Consider DAXPY: for (i = 0; i < N; ++i)  
\[ y[i] = a \times x[i] + y[i] + x[i] \times x[i] \]

For each “i”: 2 addition, 2 multiplication
2 loads of 8 bytes each
1 store

Execution on BlueGene/Q (Peak 204.8 GFLOP/node):

Performance estimates:
AI = \frac{4}{3 \times 8} = \frac{1}{6}

\frac{1}{6} < 7 \rightarrow
We are in the memory BW limited area on the roofline plot
\[ 7.11 \times \frac{1}{6} = 42.66 \]
\[ 204.8 / 42.66 = 4.8 \text{ GF/s} \]
Consider: for (i = 0; i < N; ++i)  
\[ y[i] = a \times x[i] + y[i] + x[i] \times x[i] \]

For each “i”: 2 addition, 2 multiplication  
2 loads of 8 bytes each  
1 store

Execution on BlueGene/Q (Peak 204.8 GFLOP/node):

<table>
<thead>
<tr>
<th># threads</th>
<th>Time [s]</th>
<th>GFLOPS</th>
<th>DDR traffic per node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.106501</td>
<td>0.751</td>
<td>2.906</td>
</tr>
<tr>
<td>2</td>
<td>0.053323</td>
<td>1.499</td>
<td>5.802</td>
</tr>
<tr>
<td>4</td>
<td>0.0267339</td>
<td>2.989</td>
<td>11.566</td>
</tr>
<tr>
<td>8</td>
<td>0.0176179</td>
<td>4.532</td>
<td>17.545</td>
</tr>
<tr>
<td>16</td>
<td>0.0174541</td>
<td>4.573</td>
<td>17.712</td>
</tr>
</tbody>
</table>

Performance estimates:

\[ AI = \frac{4}{3 \times 8} = \frac{1}{6} \]

\[ \frac{1}{6} < 7 \rightarrow \]

We are in the memory BW limited area on the roofline plot

\[ \frac{7.11}{\frac{1}{6}} = 42.66 \]

\[ \frac{204.8}{42.66} = 4.8 \text{ GF/s} \]
Consider for \( i = 0; i < N; ++i \) \[ y[i] = a \cdot x[i] + y[i] + x[i] \cdot x[i] + \sin(x[i]) \]

Execution on BlueGene/Q (Peak 204.8 GFLOP/node):

<table>
<thead>
<tr>
<th># threads</th>
<th>Time [s]</th>
<th>GFLOPS</th>
<th>DDR traffic per node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.615393</td>
<td>1.755</td>
<td>0.503</td>
</tr>
<tr>
<td>2</td>
<td>0.307695</td>
<td>3.51</td>
<td>1.006</td>
</tr>
<tr>
<td>4</td>
<td>0.153861</td>
<td>7.018</td>
<td>2.244</td>
</tr>
<tr>
<td>8</td>
<td>0.076983</td>
<td>14.023</td>
<td>4.02</td>
</tr>
<tr>
<td>16</td>
<td>0.0385199</td>
<td>28.008</td>
<td>8.034</td>
</tr>
<tr>
<td>32</td>
<td>0.0217798</td>
<td>49.461</td>
<td>14.202</td>
</tr>
<tr>
<td>64</td>
<td>0.018496</td>
<td>58.137</td>
<td>16.73</td>
</tr>
</tbody>
</table>
y[i] = a*x[i]+y[i]

 Loads that hit in L1 d-cache = 50.01 %
    L1P buffer = 49.98 %
    L2 cache  = 0.00 %
    DDR      = 0.01 %

We spend too much time moving data: 2.284 GF/s

y[i] = a*x[i]+y[i] + x[i]*x[i] + SIN(x[i])

 Loads that hit in L1 d-cache = 97.30 %
    L1P buffer = 2.70 %
    L2 cache  = 0.00 %
    DDR      = 0.00 %

We spend less time moving data than computing 58.137 GF/s
Examples 1 and 3

\[ y[i] = a \cdot x[i] + y[i] \]

Loads that hit in L1 d-cache = 50.01 %
L1P buffer = 49.98 %
L2 cache = 0.00 %
DDR = 0.01 %

We spend too much time moving data:
2.284 GF/s
solve time: 17.5 ms

\[ y[i] = a \cdot x[i] + y[i] + x[i] \cdot x[i] + \sin(x[i]) \]

Loads that hit in L1 d-cache = 97.30 %
L1P buffer = 2.70 %
L2 cache = 0.00 %
DDR = 0.00 %

We spend less time moving data than computing
58.137 GF/s
solve time: 18.5 ms
Consider two arrays A, and B, both have dimension of NxN

B is computed from:


Arithmetic intensity: 7 adds, 1 mul, 1 load and 1 store →

\[ AI = \frac{8}{2 \cdot 8} = \frac{1}{2} \]

Estimated performance on BG/Q: 7.11 / (1/2) = 14.22;

\[ 204.8 / 14.22 = 14.4 \text{ GF/s} \]
2D Stencil: Algorithm No. 1

#pragma omp parallel for private(row,col)

for (row = 2; row < (N-2); ++row)
    for (col = 2; col < (N-2); ++col) {
    }

We run on a single BGQ node
1 mpi rank, 64 threads

HPM info:
Total weighted GFlops = 4.922
Loads that hit in L1 d-cache = 93.05 %
    L1P buffer = 5.08 %
    L2 cache = 0.00 %
    DDR = 1.86 %
Average DDR traffic per node: ld = 13.680, st = 2.757, total = 16.437 (Bytes/cycle)

What have we done wrong?
#pragma omp parallel for private(rb,cb,row,col)

for (rb = 2; rb < N; rb = rb + row_block_size){  //ROW BLOCKING
for (cb = 2; cb < N; cb = cb + col_block_size){ // COLUMN BLOCKING

    for (row  = rb; row < MIN(N-2,rb + row_block_size+1); ++row){
        for (col  = cb; col < MIN(N-2,cb + col_block_size+1); ++col){
            B_rcb[row][col] = C*A[row][col] +
        }
    }
}

HPM info:
Total weighted GFlops = 12.264
Loads that hit in L1 d-cache = 97.69 %
  L1P buffer = 1.26 %
  L2 cache = 0.34 %
  DDR = 0.70 %
Average DDR traffic per node: \(ld = 7.599\), \(st = 6.746\), total = 14.346 (Bytes/cycle)

We estimated 14.4GF/s
We got 12.264GF/s …
Exercise No 1.

- Copy /lustre/home/ibmleopold/FOR_STUDENTS/DAXPY/ex0.c
- Compile and execute daxpy
- Use 1 to 16 threads to run the program
- Estimate performance.
- Find the crossover point.
  Calculate the location (x-coordinate) of the crossover point based on hardware
  (2-socket Intel(R) Xeon(R) CPU E5-2670 @2.6GHz node) and kernel characteristics
Exercise No 2.

- Compile and execute 2D stencil code
- Use 1 to 16 threads to run the program
- Estimate performance for 2-socket Intel(R) Xeon(R) CPU E5-2670 @2.6GHz
- Compare to the achieved performance
Questions ?
How to compile

1. ssh

2. Type
   
   MODULEPATH=/lustre/utility/modulefiles:$MODULEPATH

3. Load module
   module load icc/13.1.1

Now we can use compiler icc or icpc
The Roofline Model: Principal Components to Performance

Computation [GF/s]

Communication [GB/s]

Locality

18 cores BGQ chip