High Performance Computing: Tools and Applications

Edmond Chow
School of Computational Science and Engineering
Georgia Institute of Technology

Lecture 13

Machine balance

$$B_m = \frac{\text{memory bandwidth [words/s]}}{\text{peak performance [flops/s]}} = \frac{b_{\max}}{P_{\max}} \text{[words/flop]}$$

- Ideally, $B_m \approx 1$, but usually $B_m \ll 1$, and the trend is that B_m is further decreasing
- Example:

$$\frac{(10\ \text{GB/s})\big/(8\ \text{B/word})}{20\ \text{Gflops/s}} = 0.06\ \text{words/flop}$$

Reference: Georg Hager and Gerhard Wellein, *Introduction to High Performance Computing for Scientists and Engineers*, CRC Press, 2011.

Code balance

$$B_c = \frac{\text{data traffic [words]}}{\text{floating point ops [flops]}}$$

- $1/B_c$ is often called *computational intensity*
- if $B_c > B_m$, then code is *bandwidth bound*
- Example: what is B_c for `a[i] = b[i] + c[i]` ?

Stream benchmark

Benchmark for measuring memory bandwidth, by running a memory bandwidth bound kernel

Type	Kernel	DP words	flops	B_c
COPY	`A(:)=B(:)`	2	0	n/a
SCALE	`A(:)=s*B(:)`	2	1	2.0
ADD	`A(:)=B(:)+C(:)`	3	1	3.0
TRIAD	`A(:)=B(:)+s*C(:)`	3	2	1.5

Stream benchmark on 20 core Ivy Bridge

```
$ KMP_AFFINITY=scatter OMP_NUM_THREADS=1 ./stream
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           16978.0     0.009474     0.009424     0.009716
Scale:          18354.8     0.008746     0.008717     0.008884
Add:            18561.3     0.012942     0.012930     0.012984
Triad:          18170.9     0.013239     0.013208     0.013285

$ KMP_AFFINITY=scatter OMP_NUM_THREADS=2 ./stream
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           37638.2     0.004625     0.004251     0.005550
Scale:          40837.9     0.004232     0.003918     0.005049
Add:            39794.2     0.006415     0.006031     0.007648
Triad:          39656.2     0.006435     0.006052     0.007591

$ KMP_AFFINITY=compact OMP_NUM_THREADS=2 ./stream
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           18796.9     0.008570     0.008512     0.008935
Scale:          21479.0     0.007487     0.007449     0.007706
Add:            20745.1     0.011611     0.011569     0.011808
Triad:          20746.8     0.011591     0.011568     0.011615
```

Stream benchmark on 20 core Ivy Bridge

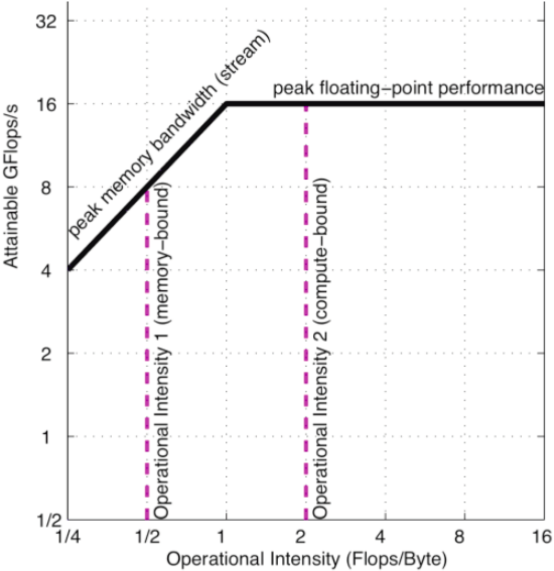
```
$ KMP_AFFINITY=compact OMP_NUM_THREADS=40 ./stream
Function     Best Rate MB/s  Avg time     Min time     Max time
Copy:           101128.5     0.001757     0.001582     0.002479
Scale:           82090.4     0.001972     0.001949     0.002020
Add:            103808.7     0.002386     0.002312     0.002505
Triad:          105362.5     0.002353     0.002278     0.002442

$ OMP_NUM_THREADS=40 ./stream
Function     Best Rate MB/s  Avg time     Min time     Max time
Copy:            78970.2     0.003858     0.002026     0.009765
Scale:           58436.8     0.004413     0.002738     0.009999
Add:             67324.3     0.005069     0.003565     0.010109
Triad:           68334.3     0.004694     0.003512     0.010124
```

Stream benchmark on Intel Xeon Phi KNC

```
Function    Best Rate MB/s  Avg time     Min time     Max time
Copy:           141341.3     0.001213     0.001132     0.001635
Scale:          117384.8     0.001502     0.001363     0.002200
Add:            132818.7     0.001883     0.001807     0.002088
Triad:          132156.1     0.001876     0.001816     0.002128
```

At what number of cores is the memory bandwidth maxed out?

Roofline model



Williams, Waterman, and Patterson, *Comm ACM*, Vol. 52, No. 4, 2009.

Computational intensity of linear algebra operations

Assuming $n \times n$ matrices and $n \times 1$ vectors, what is the number of floating point operations per memory access for:

- Vector addition
- Matrix-vector multiplication
- Matrix multiplication

Computational intensity of linear algebra operations

		flops	words	$1/B_c$
daxpy	$y = \alpha x + y$	$2n$	$3n+1$	2/3
dgemv	$y = Ax + y$	$2n^2$	n^2+3n	2
dgemm	$C = AB + C$	$2n^3$	$4n^2$	$n/2$

BLAS: Basic Linear Algebra Subroutines

These operations are available in the BLAS library

- Level 1: **Vector operations**, scale, saxpy, dot product, norms
- Level 2: **Matrix-vector operations**, sgemv (matrix-vector), rank 1 updates, rank 2 updates, triangular matvecs, triangular solves
- Level 3: **Matrix-matrix operations**, matrix-matrix product, rank-k updates, triangular solves with multiple rhs

Strive to build algorithms with higher level BLAS.

Optimized implementations for each platform

- various vendors (ACML, ESSL, Intel MKL)
- GotoBLAS (up to Nehalem), OpenBLAS
- ATLAS (autotuning)
- Reference BLAS from netlib (not optimized)