What is Performance Modeling?

• Calculate the cost of simple operations
  • Floprate
  • Reading from / writing to memory
• Can use these simple measurements to analyze the cost of a program
Why do we care?

• Want to analyze very small components of program

• Try running a program multiple times. Does it take the same amount of time each run?

  • This is amplified when we get to parallel systems

• We want to analyze programs at larger scales, which may take a very long time to run
Simple Performance Model

This model says the cost of a program is dependent on memory access and flops.

\[ T = \beta s + \sigma t \]
Simple Performance Model

\[ T = \beta s + \sigma t \]

- \( \beta \): Inverse memory access rate
- \( s \): Number of bytes read from memory
- \( \sigma \): Inverse flop rate
- \( t \): Number of Flops

Let's assume memory can be accessed at a rate of 2 Gbytes/sec and the flop rate is 4 GFlops/sec.

How long does it take to do matrix-matrix multiplication when all matrices are 100x100?
What’s inaccurate about this model?

• Think about previous lectures

• Take a minute and see if you can figure out what would improve this model.

• Can you think of another term to add to the model that would make it more accurate?
Yep, Caches!

\[ T = \beta_M S_M + \beta_C S_C + \sigma t \]

This model says the cost of a program is dependent on accesses to main memory, accesses to cache, and flops.

Unfortunately, this is getting more difficult to model. How do we know how many accesses are to main memory vs how many are to cache?
Could keep adding variables

What about memory latency?

\[ T = \alpha_M n_M + \beta_M s_M + \beta_c s_c + \sigma t \]
Could keep adding variables

What about memory latency?

\[
T = \alpha_M n_M + \beta_M s_M + \beta_c s_c + \sigma t
\]

- \(\alpha_M\): Memory latency
- \(n_M\): Number of accesses to main memory
- \(\beta_M\): Inverse main memory access rate
- \(s_M\): Number of bytes read from main memory
- \(\beta_c\): Inverse cache access rate
- \(s_c\): Number of bytes read from cache
- \(\sigma\): Floprate
- \(t\): Number of floating point operations

How many times are we accessing memory?
Do we have prefetching that hides this cost?
• This performance model is becoming more like a simulation

• We only care about modeling the majority of the cost

• So, if we are looking to model the performance of matrix-matrix multiplication, we care about memory accesses, cache accesses, and floating point operations (and can expect floating point operations to dominate the cost for any reasonable implementation).
**Caches**

\[ T = \beta_M S_M + \beta_C S_C + \sigma t \]

Inverse main memory access rate
Number of bytes read from main memory
Inverse cache access rate
Number of bytes read from cache
Floprate
Number of floating point operations

**How do we determine cache hits / cache misses**

Idea: Run cache grind (or similar profiler) on program
Get an estimate of cache hit ratio

Most simple yet, estimate the cache reuse ratio
After a value is read from memory, how many times do you reuse it?
Caches

\[ T = \beta_M n_M + \beta_c n_c + \sigma t \]

Let’s assume memory can be accessed at a rate of 2 Gbytes/sec, cache can be accessed at 10 GBytes/sec, and the flop rate is 4 GFlops/sec.

For i=0 to 1000:
For j=0 to 1000:
For k=0 to 1000:
A[i][k] = B[i][j] * C[j][k]

\( \beta_M \) inverse main memory access rate

\( n_M \) number of bytes read from main memory

\( \beta_c \) inverse cache access rate

\( n_c \) number of bytes read from cache

\( \sigma \) flop rate

\( t \) number of floating point operations
Caches

\[ T = \beta_M n_M + \beta_c n_c + \sigma t \]

Let's assume memory can be accessed at a rate of 2 Gbytes/sec, cache can be accessed at 10 GBytes/sec, and the flop rate is 4 GFlops/sec.

For i=0 to 1000:
  For j=0 to 1000:
    b_val = B[i][j]
  For k=0 to 1000:
    A[i][k] = b_val * C[j][k]

Reusing 1000 times in a row
This is in register

For i=0 to 1000:
  For j=0 to 1000:
    b_val = B[i][j]
  For k=0 to 1000:
    A[i][k] = b_val * C[j][k]

A[i][k] is reused for all j’s
So also use each 1000 times
L1 Cache accesses (after first)

C[j][k] is accessed in order, from main memory

Inverse main memory access rate
Number of bytes read from main memory

Inverse cache access rate
Number of bytes read from cache

Flop rate

Number of floating point operations
Let's assume memory can be accessed at a rate of 2 Gbytes/sec, cache can be accessed at 10 GBytes/sec, and the flop rate is 4 GFlops/sec.

For $i=0$ to $10,000$:
For $j=0$ to $10,000$:
\[ b_{\text{val}} = B[i][j] \]
For $k=0$ to $10,000$:
\[ A[i][k] = b_{\text{val}} \times C[j][k] \]

A[i][k] is reused for all j's
So also use each 10,000 times
10,000 doubles don't fit in my L1 Cache

C[j][k] is accessed in order, from main memory

Inverse main memory access rate
Number of bytes read from main memory

Inverse cache access rate
Number of bytes read from cache

Flop rate
Number of floating point operations