GPU Microarchitecture Note Set 1c—Program Needs and System Cap

Our Goal:

Determine whether we should be satisfied...
... with the performance of our program...
... on our system.

We are not satisfied when the program runs more slowly then we expect.

Knowing when to be satisfied is a key skill.

The Idea:

- Estimate the *computation needs* of our problem, algorithm, or program.

  For example, 10^{20} floating-point operations.

- Determine the *computation capabilities* of our system.

  For example, 10^{17} FLOPS.

- Use these to estimate execution time.

  For example, $10^{20}/10^{17} = 1000\,\mathrm{s}$.

If the estimate does not match the measured value, find the cause.

Computation Needs

These refer to a program, algorithm, or problem.

Determined by analyzing a problem, algorithm, or program.

Common Computation Needs:

- Floating-Point Operations
- Instruction Count (Number of executed instructions)
- Data Transfer (bytes read from and written to memory).

Computation Capabilities

These refer to the capabilities of a system (CPU, GPU, etc.)

Common Computation Capabilities

Floating Point Bandwidth (Execution Rate) - TFLOPS

Number of floating-point operations divided by amount of time.

Instruction Bandwidth (Execution Rate) - IPC

Number of executed instructions divided by amount of time.

Data Transfer Bandwidth - GB/s

Number of bytes crossing some line, divided by amount of time.

Computation Need: Number of Floating Point Operations (FLOPS)

Commonly used for scientific programs...
... because such operations are considered essential (can't be avoided) and...
... because of the historic high cost of FP hardware.

Example: fragment of some scientific code.

```
double *x, *a, *b, *c;
for ( i=0; i<1e9; i++ ) x[i] = a[i] + b[i] * c[i];
```

Uses 10^9 FP multiply/add operations.

Consider a system capable of 100 GFLOPS (10^{11} FLOPS).

Execution time bound is 10 ms.

If measured execution time was 100 ms we would not be satisfied.

Computation Need: Data Transfer

Rough Definition

The amount of data that must cross a boundary.

The boundary may be:

Between the chip and external memory. (This is common.)

Between the L2 and L1 caches.

Between the L1 cache and the computation core.

Used for all kinds of programs.

Easy to measure when each data item read exactly once ...
... making the analysis very simple.

If data reused, need to account for cache and scratchpad performance.

Data Transfer Simplest Example:

```
double *x, *a, *b, *c;
for ( i=0; i<1e9; i++ ) x[i] = a[i] + b[i] * c[i];
```

The size of a double is 8 bytes.

In an iteration, one element each of `a`, `b`, and `c` are accessed...
... and one element of `x` is written.

Here each element is read exactly once.

Total data transfer needs: $10^9 \times 8 \times 4 = 32\,\text{GB}$.

Suppose system can transfer $30\,\text{GB/s}$.

Performance bound: $32/30 = 1.067\,\text{ms}$.

Data Transfer Example:

```
double *x, *a, *b;
for ( i=1; i<1e9-1; i++ ) x[i] = a[i-1] * l + a[i] + a[i+1] * r;
```

The size of a double is 8 bytes.

In an iteration, three elements of `a` are accessed...
... two were accessed in a prior iteration...
... and so shouldn't need to be read from memory...
... leaving one new element of `a` accessed per iteration.

Total data transfer needs: $10^9 \times 8 \times 2 = 16\,\text{GB}$.

Suppose system can transfer $30\,\text{GB/s}$.

Performance bound: $16/30 = 533\,\text{ms}$.

Data Transfer Example, Continued.

Suppose measured execution time were just 1.07 s. We are not satisfied.

Maybe we were wrong about each element of `a` being read once.

Maybe we should look at number of FP operations.

Maybe we should look at number of instructions.

Computation Needs, Matrix Multiplication

Important because matrix multiplication is a common operation.

Interesting because data transfer amount depends on . . .
. . . the algorithm and the amount of storage on the compute side.

Code for multiplying two $d \times d$ matrices.

```
for ( int r=0; r<d; r++ ) for ( int c=0; c<d; c++ ) for ( int k=0; k<d; k++ )
  c[r,c] += a[r,k] * b[k,c];   // This line executed d^3 times.
```

Computation Needs

FP Operations: d^3 multiply/add operations.

Data transfer upper bound: $4d^3$ elements . . .
. . . when each iteration reads three values, `a`, `b`, and `c` and writes one, `c`.

Data transfer lower bound: $3d^2$ elements . . .
. . . when `a` and `b` each cross the boundary once (from memory to the compute hardware side) . . .
. . . and `c` crosses once (from the compute hardware side to memory).

Implications of Large Gap Between Upper- and Lower-Bound on Data

Matrix Multiplication Computation Needs (from previous slide)

FP Operations: d^3 multiply/add operations.

Data transfer upper bound: $4d^3$ elements.

Data transfer lower bound: $3d^2$ elements.

The amount of data transfer depends on the algorithm (and hardware).

If the amount of data transfer is too high ...
... floating-point units will sit idle part of the time.

We hate idle FP units, so we re-do our algorithm so less data is transferred ...
... and now FP units are busy almost all the time.

At this point reducing data transfer even further ...
... has no benefit for computation time ...
... though we might be consuming less energy.

A Numerical Example

Matrix size $d = 7722$.

FP Ops: $d^3 = 460.457 \times 10^9$ multiply/adds.

Data transfer lower bound: $3d^2 4\,\mathrm{B} = 0.716\,\mathrm{GB}$.

Data transfer upper bound: $4d^3 4\,\mathrm{B} = 7367.317\,\mathrm{GB}$.

RTX 4090 Capabilities

Single-precision FP Bandwidth: $\Theta_{\mathrm{IF}} = 41288\,\mathrm{SP\,GFLOP/s}$.

Off-Chip Data Bandwidth: $\Theta_{\mathrm{M}} = 1000\,\mathrm{GB/s}$.

Execution Limits

FP Ops: $d^3/\Theta_{\mathrm{IF}} = 11.152\,\mathrm{ms}$.

Data, lower bound: $3d^2 4\,\mathrm{B}/\Theta_{\mathrm{M}} = 0.716\,\mathrm{ms}$

Data, upper bound: $4d^3 4\,\mathrm{B}/\Theta_{\mathrm{M}} = 7367.317\,\mathrm{ms}$

Implications: Need to get data transfer closer to lower bound.

How close? Solve the equation below for D, data amount:

$$\frac{D}{\Theta_{\mathrm{M}}} = \frac{d^3}{\Theta_{\mathrm{IF}}} \quad \Rightarrow \quad D = d^3 \frac{\Theta_{\mathrm{M}}}{\Theta_{\mathrm{IF}}}$$

Plugging in the numbers: $D = 7722^3\,\mathrm{FLOP}\frac{1000\,\mathrm{GB/s}}{41288\,\mathrm{SP\ GFLOP/s}} = 11.152\,\mathrm{GB}$

An easier-to-understand number is $\frac{D}{2d^2}$, the number of times each input matrix element needs to be reused.

That works out to $\frac{11.152\,\mathrm{GB}}{2\times 7722^2\times 4\,\mathrm{B}} = 23.378$.

Tiled Matrix Multiplication

Consider

Matrix multiplication is done on a CPU (similar analysis for GPU).

CPU has an C-element cache.

Cache-to-CPU bandwidth: very large.

Cache-to-memory bandwidth: Θ_{M}.

What is the largest matrix size in which ...
... each matrix element is read or written from memory once?

```
for ( int r=0; r<d; r++ ) for ( int k=0; k<d; k++ ) for ( int c=0; c<d; c++ )
  c[r,c] += a[r,k] * b[k,c];   // This line executed d^3 times.
```

In code above we need cache storage for matrices `c` and `b`, each d^2 elements.

Only need cache storage (or registers) for one element of `a`.

So, largest size is $2d^2 + 1 = C$ or $d = \sqrt{(C-1)/2}$.

What if we need to multiply larger matrices?

Matrix Multiplication Lower Bounds

Data lower bound for M elements of storage (Hong and Kung [2, 1]): $M \left\lfloor \frac{d^3}{\sqrt{(4M)^3}} \right\rfloor \approx d^3/\sqrt{64M}$

Computation

$O(d^{\log_2 7}) \approx O(d^{2.807})$ FP ops practically possible. (Strassen's Algorithm [3].)

Instruction Count and Execution Rate

Considered a less fundamental bound than FLOPS and data transfer.

Number of instructions determined by many factors:

- How program is written.
- Quality of compiler.
- Instruction set of processor.

Performance Estimation

With above needs and capabilities can compute performance bounds.

Actual performance will be no greater than the smallest of these.

There's another measure that prevents this peak performance from being achieved.

Latency

Latency:
The time needed to do something from start to finish.

In discussion of GPUs, latency often refers...
... to the execution time of a single instruction or of a single thread.

A processor is called *latency oriented* ...
... if it has low latency (runs a single thread quickly).

References:

[1] Ballard, G., Demmel, J., Holtz, O., and Schwartz, O. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications 32*, 3 (2011), 866–901. https://doi.org/10.1137/090769156.

[2] Hong, J.-W., and Kung, H. T. I/O complexity: The red-blue pebble game. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1981), STOC '81, ACM, pp. 326–333. http://doi.acm.org/10.1145/800076.802486.

[3] Strassen, V. Gaussian elimination is not optimal. *Numer. Math. 13*, 4 (Aug 1969), 354356. https://doi.org/10.1007/BF02165411.