Housekeeping

- Your goal today: appreciate the many subtleties and dimensionalities of performance
  Digested from three 18-447 lectures
- Notices
  - Handout #3: lab 1, due noon, 9/22
  - ZedBoard ready for pick up (see Handout #3a)
  - Recitation starts this week, Wed 4:30~5:20
- Readings (skim for background)
The ZedBoard Kit

- ECE is loaning to you
  - 1x Avnet ZedBoard 7020 baseboard
  - 1x 12 V power supply
  - 1x 4-GB SD Card (don’t need it right away)
  - 2x Micro-USB cable

- You will treat it like your grade depended on it
- You will return all of it in perfect condition, or else

Looking Ahead

- Lab 1 (wk3/4): get cozy with Vivado
  - most important: learn logic analyzer and eclipse debugger
- Lab 2 (wk5/6): meet Vivado HLS
  - most important: decide if you would use it
- Lab 3 (wk7/8): hands-on with AFU
  - most important: have confidence it can work
- Project . . .
Performance is about time

- To the first order, performance $\propto 1 / \text{time}$

- Two very different kinds of performance!!
  - latency = time between start and finish of a task
  - throughput = number of tasks finished in a given unit of time (a rate measure)

- Either way, shorter the time, higher the performance, but . . .

Throughput $\neq 1$/Latency

- If it takes $T$ sec to do $N$ tasks, throughput=$N/T$; latency=$T/N$?
- If it takes $t$ sec to do 1 task, latency=$t$; throughput=$1/t$?
- When there is concurrency, throughput$\neq 1$/latency

- Optimizations can tradeoff one for the other (think bus vs F1 race car)
Throughput ≠ Throughput

• Throughput becomes a function of $N$ when there is a non-recurring start-up cost (aka overhead)
• E.g., DMA transfer on a bus
  – bus throughput$_{raw}$ = 1 Byte / ($10^{-9}$ sec)
  – $10^{-6}$ sec to setup a DMA
  – throughput$_{effective}$ to send 1B, 1KB, 1MB, 1GB?
• For start-up-time = $t_s$ and throughput$_{raw}$ = $1/t_1$
  – throughput$_{effective}$ = $N / (t_s + N \cdot t_1)$
  – if $t_s >> N \cdot t_1$, throughput$_{effective}$ ≈ $N/t_s$
  – if $t_s << N \cdot t_1$, throughput$_{effective}$ ≈ $1/t_1$
  
  we say $t_s$ is “amortized” in the latter case

Latency ≠ Latency

• What are you doing during the latency period?
• Latency = hands-on time + hands-off time
• In the DMA example
  – CPU is busy for the $t_s$ to setup the DMA
  – CPU has to wait $N \cdot t_1$ for DMA to complete
  – CPU could be doing something else during $N \cdot t_1$ to “hide” that latency
Relative Performance

• Pop Quiz: if \( X \) is 50% slower than \( Y \) and latency_{X}=1.0s, what is latency_{Y}?

  – Case 1: \( L_{Y} = 0.5s \) since \( L_{Y}/L_{X}=0.5 \)

  – Case 2: \( L_{Y} = 0.66666s \) since \( L_{X}/L_{Y}=1.5 \)

  English language is imprecise

Fixing the language, a la H&P

• “\( X \) is \( n \) times faster than \( Y \)” means

  \[ n = \frac{\text{performance}_{X}}{\text{performance}_{Y}} = \frac{\text{throughput}_{X}}{\text{throughput}_{Y}} = \frac{\text{latency}_{Y}}{\text{latency}_{X}} = \text{“speedup” from } Y \text{ to } X \]

• “\( X \) is \( m\% \) faster than \( Y \)” means

  \[ 1+m/100 = \frac{\text{Performance}_{X}}{\text{Performance}_{Y}} \]

• Delete “slower” from your dictionary
  – for case 1 say, “\( Y \) is 100% faster than \( X \) in latency”
  – for case 2 say, “\( Y \) is 50% faster than \( X \) in latency”
**Faster!=Faster**

- Given two designs X and Y,
  - X may be m% faster than Y on input A
  - X may be n% (where m!=n) faster than Y on input B
  - Y may be k% faster than X on input C
- Which is faster and by how much?
  - depends on which input(s) you care about
  - if multiple, also depend on relative importance
- Many ways to summarize performance into a scalar metric to simplify comparison
  - more wrong ways than right
  - when in doubt, present the complete story
  - go read the H&P chapter on performance

**Multi-Dimensional Optimizations**

- HW design has many optimization dimensions
  - by area, by resource type utilization
  - performance and latency
  - power and energy
  - complexity, risk, social factors . . .
- Cannot optimize individual metrics without considering tradeoff between them, e.g.,
  - reasonable to spend more power for performance
  - converse also true (lower perf. for less power)
  - but never more power for lower performance
Pareto Optimality (2D example)

All points on front are optimal (can’t do better)

How to select between them?

Application-Defined Composite Metrics

- Define scalar function to reflect desiderata---incorporate dimensions and their relationships
- E.g., energy-delay-(cost) product
  - smaller the better
  - can’t cheat by minimizing one ignoring others
  - does it have to have a physical meaning??
- Floors and ceilings
  - real-life designs more often about good enough than optimal
  - e.g., meet a perf. floor under a power(cost)-ceiling
    (minimize design time, i.e., stop when you get there)
Power $\neq$ Energy
(do not confuse them)

Power = Energy / time

- Energy (Joule) dissipated as heat when “charge” flow from VDD to GND
  - takes a certain amount of energy per operation, e.g., addition, reg read/write, (dis)charge a node
  - to the first order, energy $\propto$ work
- Power (Watt=Joule/s) is rate of energy dissipation
  - more op/sec then more Joules/sec
  - to the first order, power $\propto$ performance

It is all very easy if performance $\propto$ frequency
Power=$\frac{1}{2}CV^2f$
Power and Performance not Separable

- Easy to minimize power if don’t care about performance
- Expect superlinear increase in power to increase performance
  - slower design is simpler
  - lower frequency needs lower voltage
- Corollary: Lower perf also use lower J/op (=slope from origin)

All in all, slower is more energy/power efficient

---

Slower could use more energy (if done inefficiently)

- Devices leak charge even when doing no ops
  - so called leakage current (I_{leakage})
  - power_{total} = switching power + static power
    \[ = (J/op)_{perf} \times (#op/runtime) + (VDD \times I_{leakage}) \]
  - energy_{total} = (J/op)_{perf} \times #op + (VDD \times I_{leakage} \times runtime)
- Slower reduces power but could increase energy
Perf/Watt $\neq$ Perf/Watt

- Perf/Watt is a normalized measure
  - hides the scale of problem and platform
  - recall, $\text{Watt} \propto \text{perf}^k$ for some $k>1$

- 10 GFLOPS/Watt at 1W is a very different design problem than at 1KW or 1MW or 1GW
  - say 10 GFLOPS/Watt on a <GPGPU,problem>
  - now take 1000 GPGPUs to the same problem
  - realized perf is $< 1000x$ (less than perfect parallelism)
  - required power $> 1000x$ (energy to move data & heat)

In general be careful with normalized metrics
Parallelism Defined

- $T_1$ (work measured in time):
  - time to do work with 1 PE
- $T_\infty$ (critical path):
  - time to do work with infinite PEs
  - $T_\infty$ bounded by dataflow dependence
- Average parallelism:
  $$P_{\text{avg}} = \frac{T_1}{T_\infty}$$

- For a system with $p$ PEs
  $$T_p \geq \max\{ \frac{T_1}{p}, \frac{T_\infty}{p} \}$$
- When $P_{\text{avg}} \gg p$
  $$T_p \approx \frac{T_1}{p}, \text{ aka "linear speedup"}$$

Linear Parallel Speedup

- Ideally, parallel speedup is linear with $p$
  $$\text{speedup} = \frac{\text{runtime}_{\text{sequential}}}{\text{runtime}_{\text{parallel}}}$$

- Lower runtime
  - $\propto 1/p$
- Higher speedup
Linear Speedup! = Linear Speedup

How could this be?

It could be worse .......

limited scalability, $P_{avg} < p$
Parallelization Overhead

- Best parallel and seq. algo. need not be the same
  - best parallel algo. often worse at $p=1$
  - if $\text{runtime}_{\text{parallel}}@p=1 = K\cdot\text{runtime}_{\text{sequential}}$ then best-case speedup = $p/K$

- Communication between PEs not instantaneous
  - extra time for the act of sending or receiving data as if adding more work ($T_1$)
  - extra time waiting for data to travel between PEs as if adding critical path ($T_\infty$)

If overhead grows with $P$, speedup can even fall

Parallelization not just about speedup

- For a given functionality, non-linear tradeoff between power and performance
  - slower design is simpler
  - lower frequency needs lower voltage

$\Rightarrow$ For the same throughput, replacing 1 module by 2 half-as-fast reduces total power and energy

Better to replace 1 of this by 2 of these; or $N$ of these

Good hardware designs derive performance from parallelism
**Arithmetic Intensity**

- An algorithm has a cost in terms of operation count
  - \( \text{runtime}_{\text{compute-bound}} = \frac{\# \text{ operations}}{\text{FLOPS}} \)
- An algorithm also has a cost in terms of number of bytes communicated (ld/st or send/receive)
  - \( \text{runtime}_{\text{BW-bound}} = \frac{\# \text{ bytes}}{\text{BW}} \)
- Which one dominates depends on
  - ratio of FLOPS and BW of platform
  - ratio of ops and bytes of algorithm
- Average Arithmetic Intensity (AI)
  - how many ops performed per byte accessed
  - \( \frac{\# \text{ operations}}{\# \text{ bytes}} \)

**Roofline Performance Model**

[Williams&Patterson, 2006]

Attained Performance of a system (op/sec)

\[
\text{perf} = \min(\text{FLOPS}, \text{AI} \cdot \text{BW})
\]

\[
\text{runtime} > \max\left( \frac{\# \text{ op}}{\text{FLOPS}}, \frac{\# \text{ byte}}{\text{BW}} \right)
\]

\[
> \# \text{ op} \cdot \max(1/\text{FLOPS}, 1/(\text{AI} \cdot \text{BW}))
\]
Simple AI Example: MMM

```c
for(i=0; i<N; i++)
    for(j=0; j<N; j++)
        for(k=0; k<N; k++)
            C[i][j]+=A[i][k]*B[k][j];
```

- $N^2$ data-parallel dot-product's
- Assume $N$ is large s.t. 1 row/col too large for on-chip
- Operation count: $N^3$ float-mult and $N^3$ float-add
- External memory access (assume 4-byte floats)
  - $2N^3$ 4-byte reads (of $A$ and $B$) from DRAM
  - $\ldots N^2$ 4-byte writes (of $C$) to DRAM
- Arithmetic Intensity $\approx 2N^3/(4\cdot 2N^3)=1/4$

GTX1080: 8 TFLOPS vs 320GByte/sec

---

Less Simple AI Example: MMM

```c
for(i0=0; i0<N; i0+=Nb)
    for(j0=0; j0<N; j0+=Nb)
        for(k0=0; k0<N; k0+=Nb) {
            for(i=i0;i<i0+Nb;i++)
                for(j=j0;j<j0+Nb;j++)
                    for(k=k0;k<k0+Nb;k++)
                        C[i][j]+=A[i][k]*B[k][j];
        }
```

- Imagine a $\lfloor N/N_b \rfloor \times \lfloor N/N_b \rfloor$ MATRIX of $N_b \times N_b$ matrices
  - inner-triple is straightforward matrix-matrix mult
  - outer-triple is MATRIX-MATRIX mult
- To improve AI, hold $N_b \times N_b$ sub-matrices on-chip for data-reuse
  need to copy block (not shown)
**AI of blocked MMM Kernel ($N_b \times N_b$)**

```c
for(i=i0;i<i0+Nb;i++)
    for(j=j0;j<j0+Nb;j++) {
        t=C[i][j];
        for(k=k0;k<k0+Nb;k++)
            t+=A[i][k]*B[k][j];
        C[i][j]=t;
    }
```

- Operation count: $N_b^3$ float-mul and $N_b^3$ float-add
- When $A$, $B$ fit in scratchpad ($2 \times N_b^2 \times 4$ bytes)
  - $2N_b^3$ 4-byte on-chip reads ($A$, $B$) (fast)
  - $3N_b^2$ 4-byte off-chip DRAM read $A$, $B$, $C$ (slow)
  - $N_b^2$ 4-byte off-chip DRAM writeback $C$ (slow)
- Arithmetic Intensity = $2N_b^3/(4 \times 4N_b^2) = N_b/8$

---

**AI and Scaling**

- AI is a function of the algorithm and the problem size
- Higher AI means more work per communication and therefore easier to parallelize and to scale

[Figure 6.17, Computer Organization and Design]
**Strong vs Weak Scaling**

- **Strong Scaling**: $T_1$ (work) remains constant
  - i.e., improve same workload with more PEs
  - hard to maintain linearity as $p$ grows toward $p_{avg}$
- **Weak Scaling**: $T_1' = p \cdot T_1$
  - i.e., improve larger workload with more PEs
  - $S_p = \frac{\text{runtime}_{\text{sequential}}(p \cdot T_1)}{\text{runtime}_{\text{parallel}}(p \cdot T_1)}$
    - is this harder or easier?
    - ans: depends on how $T_\infty'$ scales with $T_1'$ and how AI scales with $T_1'$

**Amdahl’s Law**

- If only a fraction $f$ is improved by a factor of $s$
  
  \[
  \begin{array}{c}
  \text{time}_{\text{parallelized}} \\
  (1 - f) \quad f/s
  \end{array}
  \]
  \[
  \begin{array}{c}
  \text{time}_{\text{sequential}} \\
  (1 - f) \quad f
  \end{array}
  \]

  \[
  \text{time}_{\text{parallel}} = \text{time}_{\text{sequential}} \cdot \left( (1-f) + \frac{f}{s} \right)
  \]

  \[
  \text{speedup} = \frac{1}{(1-f) + \frac{f}{s}}
  \]

  - if $f$ is small, $s$ doesn’t matter
  - even when $f$ is large, diminishing return on $s$;
    eventually “$1-f$” dominates
Parting Thoughts

• Need to understand performance to get performance!
• HW/FPGA design involve many dimensions (each one nuanced)
  – optimizations often involve tradeoff
  – over simplifying is dangerous and misleading
  – must understand application needs

  power and energy is first-class

• Real-life designs have non-technical requirements