Data Structures & Algorithms for Geometry

- Agenda:
  - Memory hierarchy
  - Optimization background
  - Hands-on with VTune

Memory Hierarchy

- Computer memory is made up from a hierarchy of memory types
  - Faster memory is smaller and closer to the processor core
- Four primary levels in the hierarchy:
  - registers
  - L1 / L2 / L3 cache
  - main memory
  - disk

Data Transfer Performance

- Two primary factors affect data transfer performance:
  - Bandwidth: amount of data transfered per unit time
  - Latency: elapsed time from a data request to the arrival of the first bits
- In most cases, a balance is desired
  - High latency *can* nullify benefits of high bandwidth

> "Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway."
>
> -- Andrew S. Tanenbaum

Latency vs. Bandwidth

- Latency can be masked by using more bandwidth
  - Guess what data might be needed, and prefetch it
  - Mis-predictions waste bandwidth, but add essentially zero extra latency
  - Correct predictions can eliminate nearly all latency
- Most CPUs and memory controllers do *some* of this automatically
  - All modern CPUs include special instructions to request a memory prefetch

Latency Survey

- Latencies for common current generation processors:
  - registers: 0 or 1 clock cycle access, 512 bytes
  - L1 cache: 3 or 4 clock cycle access, 32kB - 128kB
  - L2 cache: 14 – 32 clock cycle access, 1MB – 12MB
  - main memory: 150 – 300 clock cycle access, 1GB – 16GB
  - disk: ~20 *million* clock cycle access

Locality of Reference

- All caching is based on one principle: related storage locations are frequently accessed
- Three forms:
  - Temporal: if a given location is accessed once, it will likely be accessed again
  - Spatial: if a given location is accessed, locations near it will likely be accessed
  - Sequential: if location *N* is accessed, location *N*+1 will likely be accessed

Cache Lines

- Locations are grouped into cache *lines*
  - Lines are typically 16, 32, or 64 bytes
  - When location *N* is accessed, the entire line containing it is fetched
    - Spatial locality

Cache Replacement

- As data is fetched into the cache, it will eventually become full
  - When this occurs, data must be discarded to make room for new data
- Nearly all caches use a least-recently-used (LRU) replacement policy
  - The data least recently used is evicted
    - Temporal locality

What does this mean for us?

- Try to organize data storage and data access to maximize cache usage
  - Store structure elements that will be accessed together within a single cache line
  - Avoid use of pointer-linked structures when possible
  - Store blocks of data that will be accessed together in a single block
  - Use prefetching instructions to pull data in *before* it is needed

Array of Structures

- Consider this edge structure:

```
struct edge {
    edge      *cw[2];
    edge      *ccw[2];
    faces     *f[2];
    vertex    *v[2];
    edge_link *owner[2];
};
```

- Data grouping is bad
  - `cw` and `ccw` will be used for edge traversal, but `vertex` is unlikely to be used at that time
  - `vertex` “pollutes” the cache and wastes space

Structure of Arrays

- Consider this edge structure:

```
struct edge_table {
    unsigned  *cw_list;
    unsigned  *ccw_list;
    faces     **f_list;
    vertex    **v_list;
    edge_link **owner_list;
};
```

- Data grouping is much better!
  - All values `cw` are stored together
  - `vertex` data no longer pollutes the cache while performing edge traversal

References

Tanenbaum, Andrew S. Computer Networks. New Jersey: Prentice-Hall, 1983. ISBN 0-13-349945-6.

Denning, P. J. 2005. The locality principle. Communications of the ACM 48, 7 (Jul. 2005), 19-24. http://cs.gmu.edu/cne/pjd/PUBS/CACMcols/cacmJul05.pdf

van der Pas, Ruud. Memory Hierarchy in Cache-Based Systems. Sun Microsystems. 2002. http://www.sun.com/blueprints/1102/817-0742.pdf

About optimization...

"Premature optimization is the root of all evil"

-- Donald Knuth

Amdahl's Law

- Predicts the maximum possible *overall* speed up achieved by speeding up a given component

$$\frac{1}{(1-P)+\frac{P}{S}}$$

  - *P* is the percentage of the whole to be improved
  - *S* is the speed up in that portion
  - The speed up is the inverse of the time taken for the unoptimized portion plus the new time of the optimized portion

Amdahl's Law (cont.)

- If a program spends 10% of its time in one code segment, what is the maximum improvement we can get by optimizing that code?

Amdahl's Law (cont.)

- If a program spends 10% of its time in one code segment, what is the maximum improvement we can get by optimizing that code?
  - $P = 0.1$, $S = \infty$

$$\frac{1}{(1-P)+\frac{P}{S}} = \frac{1}{1-.1+\frac{.1}{\infty}} = \frac{1}{1-.1+0} = \frac{1}{.9} = 1.111...$$

Amdahl's Law (cont.)

- If a program spends 10% of its time in one code segment, what is the maximum improvement we can get by optimizing that code?
  - $P = 0.1$, $S = \infty$

$$\frac{1}{(1-P)+\frac{P}{S}} = \frac{1}{1-.1+\frac{.1}{\infty}} = \frac{1}{1-.1+0} = \frac{1}{.9} = 1.111...$$

  - 1.1x improvement is probably not worth the effort

Profiling

- Profiling guides **all** optimization efforts
  - Determine where the most time is spent
  - Optimize that portion first
  - Profile again...lather, rinse, repeat...
- Determine if the code needs to be optimized
  - Is it cheaper to buy faster hardware than to pay programmers to optimize the code?
  - You have to know what is “fast enough”

References

Knuth, Donald. 1974. Structured Programming with Goto Statements. Computing Surveys 6:4, 261–301. http://pplab.snu.ac.kr/courses/adv_pl05/papers/p261-knuth.pdf

http://en.wikipedia.org/wiki/Amdahl's_law

Next week...

- Numerical and geometric robustness
- Prepare for final

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.
- VTune is a trademark of Intel Corporation.
- Other company, product, and service names may be trademarks or service marks of others.