instruction pipelining
and
pre-computation in neural networks

colin shaw
09.08.2016

1. **introduction**
    a. welcome to computer science club
    b. thank you to revunit for sponsoring
    c. what we talked about the last two times
        i. multiple layer perceptron with nice dsl in ocaml
        ii. improving matrix arithmetic computational performance
        iii. difference between functional and imperative programming
    d. what we are talking about this time
        i. computer organization and speed optimization
            1. memory organization
            2. caching methods
            3. processor pipelining
            4. what environment makes numeric code fast
            5. code design to take advantage of cache and pipelining (including ocaml)
            6. wire, silicon and the speed of light
        ii. neural network problem space reduction
            1. problem size vs training time in neural networks
            2. reducing feature space as pre-computation in neural networks
    e. why are we talking about this?
        i. need to know how large a problem can be solved in known time
        ii. need to know how how large a problem training a mlp is
    f. why does that matter?
        i. real time training
        ii. low cost
        iii. low power

2. **memory organization**
    a. alu / fpu
    b. processor registers (typically 8 - 16)
    c. l1i and l1d cache
        i. instruction and data separate
        ii. implemented per core
        iii. fabricated on processor die
        iv. sram
        v. 0 - 64kb, power of two
        vi. ≅1ns access time
    d. l2 cache
        i. blend of instruction and data
        ii. higher end fabricated per core
        iii. lower end shared between cores
        iv. fabricated on processor die
        v. sram
        vi. 256kb - 1s of mb, power of two
        vii. ≅5 ns access time
    e. l3 cache
        i. blend of instruction and data
        ii. shared between cores
        iii. higher end fabricated on processor die (sram)
        iv. lower end fabricated off processor die (dram)
        v. 1s of mb - 10s of mb, generally power of two
        vi. ≅30 ns access time
    f. l4 cache
        i. mostly commonly implemented as edram for video
        ii. focus is bandwidth rather than access time
        iii. Embedded off processor die (edram)
        iv. 10s of mb - 100s of mb, power of two
    g. main memory
        i. blend of instruction and data
        ii. shared
        iii. off processor die (dram)
        iv. 1s of gbs - 10s of gb
        v. ≅100 ns access time
        vi. 10,000+ mb/s (pc1333)
    h. solid state disk
        i. 10,000 ns access time
        ii. 500 mb/s
    i. rotational hard disk
        i. 5,000,000 ns access time
        ii. 200 mb/s

3. **cache operation**
    a. cache lines
        i. basic unit of transport from cache to processor
        ii. loading one byte in a line loads the whole line
        iii. typically 16 bytes - 128 bytes (nominally 64)
    b. cache types
        i. direct mapped
            1. each main memory location can go to exactly one cache location
            2. suffers contention in worst case
        ii. fully associative
            1. each main memory location can map to any cache location
            2. costly to implement
        iii. n-way associative
            1. compromise between direct mapped and fully associative
            2. each main memory location can go to n potential cache locations
            3. most common implementation of caching
            4. generally 2-way to 16-way associativity
            5. difference between intel and amd
    c. translation lookaside buffer (tlb)
        i. address virtualization
        ii. used as fast memory reference without computing physical page

4. **processor pipelining**
    a. key steps
        i. read
        ii. load
        iii. execute
        iv. write
    b. queueing
        i. grocery checkout analogy
    c. how to control
        i. modern processors have 12 - 24 stage pipeline
        ii. branch and memory access prediction
        iii. some assembly / compiler control
    d. multimedia and math extensions (intel)
        i. mmx
        ii. sse
        iii. avx (256 bit registers)

5. **what environment makes numeric code fast**
    a. fast memory
        i. dominated by main memory bandwidth
        ii. sequential access best
        iii. cache line alignment beneficial, especially for short sequential access
        iv. buy the fastest you can afford
    b. registers and cache
        i. enough registers for loops and intermediate values
        ii. cache alignment
        iii. prefetching and other memory optimizations
        iv. buy the largest cache you can afford
        v. pay attention to cache associativity
    c. fast processor
        i. relative core speed to memory bandwidth
        ii. generally involves pipelining and memory fetch consideration
        iii. example raspberry pi vs intel processor
        iv. buy the fastest core you can afford with the right pipelining
    d. intelligent choice of types
        i. traditional numeric options
            1. single precision (32 bit)
            2. double precision (64 bit)
        ii. more exotic numeric options
            1. long double (80 or 128 bit)
            2. half precision (16 bit)
                a. what is it
                    i. generally a storage format with single computation
                    ii. better memory bandwidth
                    iii. small cost for conversion to/from single
                    iv. worse performance if sequence < l1d cache size
                b. where is it seen in the wild
                    i. arm
                    ii. gpu
                    iii. intel generation 3 and above
        iii. which is best and why
            1. smallest sufficient for problem
            2. best memory throughput
            3. best pipelining
    e. algorithm design
        i. column-major vs row-major organization
        ii. loop unrolling (often done better by compiler)
    f. compiler considerations
        i. quality of loop unrolling
        ii. quality of memory and cache consideration (prefetch, etc.)
        iii. supported vectorization
        iv. function inlining
        v. type support (half precision, etc.)

6. **coding examples for cache and pipelining**
    a. experiments
        i. cache_line.c
            1. cache line utilization vs aggregate speed
            2. x86 cache line is 64 bytes
            3. non-optimized and optimized (e.g. -o3)
            4. optimization is cache line alignment
        ii. cache_speed.c
            1. normalized time for sequential access by size
            2. detects l1 and l2 size
        iii. pipeline.c
            1. memory lookup pipelining and compiler optimization
            2. non-optimized and optimized
        iv. pipeline2.c
            1. instruction level pipelining
            2. non-optimized and optimized
            3. compiler may optimize two adds as one in tight loops
        v. thread.c
            1. pathological cache line access patterns with threading
            2. false cache line sharing
    b. best practice patterns
        i. smallest type meeting problem criteria
        ii. tight loops
        iii. sequential accesses
        iv. function inlining
        v. compiler hinting
        vi. compiler optimization settings
        vii. compiler choice
            1. manufacturer with hand tuning
            2. general with optimizations
            3. general without optimizations
    c. practical implementation with ocaml
        i. ocaml_native.ml
            1. intrinsic array map
        ii. ocaml_no_opt_ml.ml / ocaml_no_opt_c.c
            1. array map done in c
        iii. ocaml_opt_ml.ml / ocaml_opt_c.c
            1. bigarray
            2. in-place
            3. extra compiler optimizations for single precision floating point

7. **wire, silicon and the speed of light**
    a. speed of information in wire
        i. 1 ns (1 ghz) ≅ 1 foot propagation
        ii. limitations
            1. distance from l1 cache to core at core speed
            2. distance from processor to dram distance at memory speed
    b. cmos process
        i. mosfet gate area shrinks quadratically as lithography shrinks
        ii. gate thickness single atom
            1. new gate material is new trend
            2. was silicon dioxide
            3. new materials harder harder to fab, more expensive
        iii. gate capacitance becomes limiting
            1. rc circuit of gate becomes new prevailing issue
            2. vertical wire paths new technique
                a. can make direct, euclidean distance, connections
                b. most silicon connections are manhattan distance
            3. gates as amplifiers in path to circumvent rc time
        iv. heat generation
            1. moving charges generate heat
            2. faster clock yields more heat
            3. higher density yields higher temperature rise
            4. must be at least somewhat above bandgap to work (limit)
    c. notes on lithography
        i. current production technology
            1. 14nm
                a. smaller than smallest known virus
                b. 40 times shorter than wavelength of green light
            2. 5nm by early 2020s
        ii. how it works
            1. (extreme) ultraviolet photolithography
            2. wavelength (slightly) smaller than features
            3. diffraction effects specifically used
            4. surface roughness is problematic (atomic surface is 0.23 nm)
            5. mask and etch, mask and etch...

8. **neural network problem size and training time**
    a. theory
        i. problem degrees of freedom
            1. total inputs and outputs
            2. depends on this being proper size for problem
                a. cannot be underdetermined or will have training problems
                b. cannot be overdetermined or will not not be good estimator
        ii. hidden layer effects
            1. each neuron is independent per layer
            2. each layer is independent because of nonlinear activation function
            3. hidden layer neurons are multipliers of problem dof
        iii. putting it all together
            1. $n_t = \boldsymbol{\alpha} (n_i + n_o) n_h$
    b. simplification
        i. assumptions
            1. $n_i = \nu^2$
            2. $n_o = k_1$
            3. $n_h = k_2 \nu$
        ii. estimation
            1. $n_t = \boldsymbol{\alpha} (\nu^2 + k_1) k_2 \nu$
            2. n_t bounded by $\boldsymbol{\alpha} k_2 \nu^3$
        iii. example
            1. assumptions
                a. $n_t = 2000$
                b. $\boldsymbol{\alpha} = 2$
                c. $k_2 = 1$
            2. $\nu \cong 10$

9. **pre-processing in neural networks**
    a. rationale
        i. biological imperative - retina
        ii. problem determinism
        iii. improved performance
        iv. improved understanding of problem
    b. an example for detecting sidewalk curvature
        i. create subsampled images from original feature set
        ii. threshold images to black and white
        iii. image output is zero if not enough blend of black and white
        iv. find centroid of black and white
        v. image vector between centroids
        vi. create perpendicular vector, indicating tangent of curve
        vii. image output is angle of tangent
        viii. expectation is zeros except sides of sidewalk