Advanced Topics in Numerical Analysis: High Performance Computing

Memory hierarchies

Georg Stadler Dhairya Malhotra
Courant Institute, NYU
stadler@cims.nyu.edu dm4340@nyu.edu

Spring 2019, Monday, 5:10–7:00PM, WWH #1302

Feb 4, 2019

Outline

Organization issues

Summary of last class

Memory hierarchies (single CPU)

Tools/commands

Organization issues

- If you're not in the class' Slack group yet, please let us know.

Organization issues

- If you're not in the class' Slack group yet, please let us know.
- Homework assignment #1 is posted and due next week.
  - Part I: Find examples for HPC; Hand in a PDF separately, we will post link to folder where you can put this file–everybody will have access to it.
  - Part II: Simple single-core examples. We'll improve these over the semester.

Organization issues

- If you're not in the class' Slack group yet, please let us know.
- Homework assignment #1 is posted and due next week.
  - Part I: Find examples for HPC; Hand in a PDF separately, we will post link to folder where you can put this file–everybody will have access to it.
  - Part II: Simple single-core examples. We'll improve these over the semester.
- Public course website contains an outline (which might change a bit): `https://nyu-hpc19.github.io/`
- Questions?

Plan for today

- Summary of last week's material (Moore's law, multicore, HPC overview)
- Finish example problems in OpenMP and MPI
- Memory hierarchies (caches), basic performance models, single core performance
- Code examples; *Tools*: valgrind and cachegrind

Outline

Organization issues

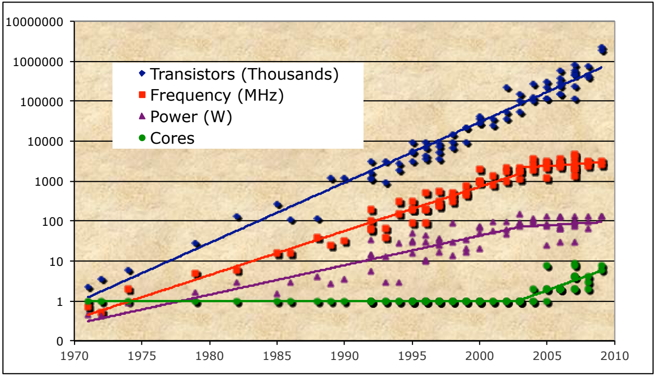
Summary of last class

Memory hierarchies (single CPU)

Tools/commands

Moore's law today

- Frequency/clock speed stopped growing in ~ 2004
- Number of cores per CPU
- Moore's law still holds
- Energy use ~bounded



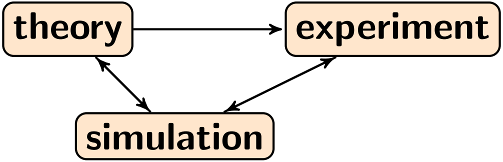
Source: CS Department, UC Berkeley.

Parallel computing $\subset$ high-performance computing

- All major vendors produce multicore chips—need to think differently about applications.
- How well can applications and algorithms exploit parallelism?
- Memory density (DRAM) grows at slower rate.
  Loading/writing to memory is slow ($\mathcal{O}(100)$ clock cycles)
- Top500 list: leading machines have $> 10^7$ processor cores, and often two different kinds of compute chips (CPUs and some kind of accelerators (e.g., GPUs)).

Do we really need larger and faster?

Simulation has become the third pillar of Science:



HPC computing used in: weather prediction, climate modeling, drug design, astrophysics, earthquake modeling, semiconductor design, crash test simulations, financial modeling, ...

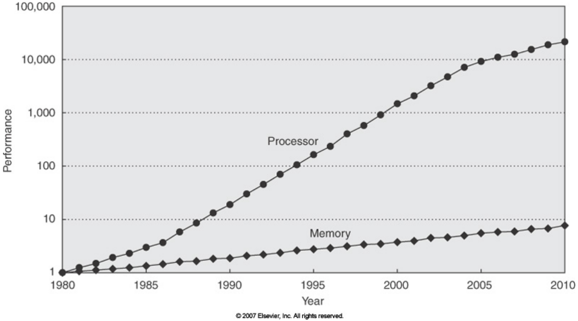
Basic CS terms recalled

- compiler: translates human code into machine language
- CPU/processor: central processing unit caries out instructions of a computer program, i.e., arithmetic/logical operations, input/output
- core: individual processing unit in a CPU, “multicore” CPU; will sometimes use “processors” in a sloppy way, and actually mean “cores”
- clock rate/frequency: indicator of speed in which instructions are performed
- floating point operation: multiplication add of two floating point numbers, usually double precision (64 bit, about 16 digits)
- peak performance: fastest theoretical flop/s
- sustained performance: flop/s in actual computation
- memory hierarchy: large memories (RAM/disc/solid state) are slow; fast memories (L1/L2/L3 cache) are small

Outline

Flop/s versus Mop/s

For many practical applications, memory access is the bottleneck, not floating point operations.



Development of memory versus processor performance.

- Most applications run at $< 10\%$ of the theoretical peak performance.
- Mostly a single core issue; on parallel computers, things become even more difficult.

Memory hierarchies

Computer architecture is complicated. We need a basic performance model.

- Processor needs to be “fed” with data to work on.
- Memory access is slow; memory hierarchies help.
- This is a single processor issue, but it’s even more important on parallel computers.

Memory hierarchies

Computer architecture is complicated. We need a basic performance model.

- Processor needs to be "fed" with data to work on.
- Memory access is slow; memory hierarchies help.
- This is a single processor issue, but it's even more important on parallel computers.

More CS terms:

- latency:

Memory hierarchies

Computer architecture is complicated. We need a basic performance model.

- Processor needs to be "fed" with data to work on.
- Memory access is slow; memory hierarchies help.
- This is a single processor issue, but it's even more important on parallel computers.

More CS terms:

- latency: time it takes to load/write data from/at a specific location in RAM to/from the CPU registers (in seconds)

Memory hierarchies

Computer architecture is complicated. We need a basic performance model.

- Processor needs to be "fed" with data to work on.
- Memory access is slow; memory hierarchies help.
- This is a single processor issue, but it's even more important on parallel computers.

More CS terms:

- latency: time it takes to load/write data from/at a specific location in RAM to/from the CPU registers (in seconds)
- bandwidth:

Memory hierarchies

Computer architecture is complicated. We need a basic performance model.

- Processor needs to be “fed” with data to work on.
- Memory access is slow; memory hierarchies help.
- This is a single processor issue, but it’s even more important on parallel computers.

More CS terms:

- latency: time it takes to load/write data from/at a specific location in RAM to/from the CPU registers (in seconds)
- bandwidth: rate at which data can be read/written (for large data); in (bytes/second);

Memory hierarchies

Computer architecture is complicated. We need a basic performance model.

- Processor needs to be "fed" with data to work on.
- Memory access is slow; memory hierarchies help.
- This is a single processor issue, but it's even more important on parallel computers.

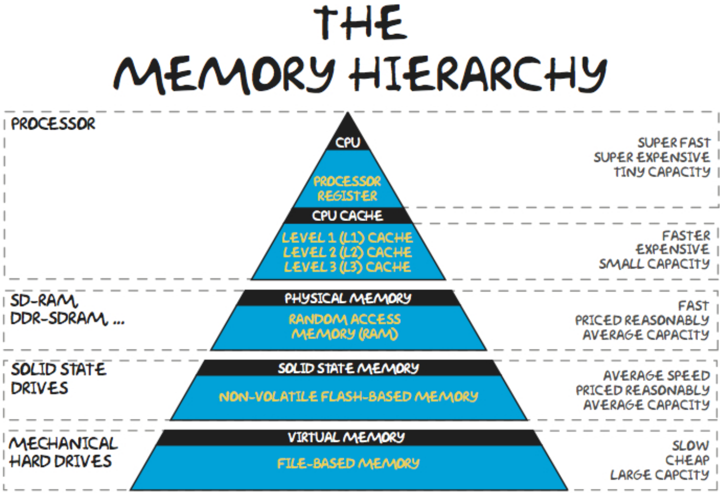
More CS terms:

- latency: time it takes to load/write data from/at a specific location in RAM to/from the CPU registers (in seconds)
- bandwidth: rate at which data can be read/written (for large data); in (bytes/second);

Bandwidth grows faster than latency.

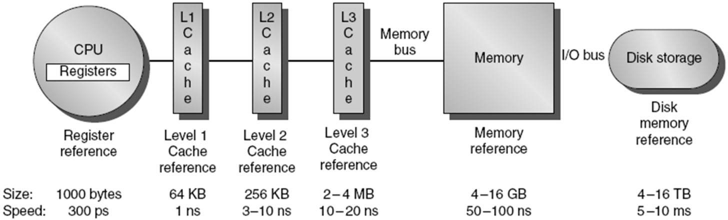
Memory hierarchies

On my Mac Book Pro: 32KB L1 Cache, 256KB L2 Cache, 3MB Cache, 8GB RAM

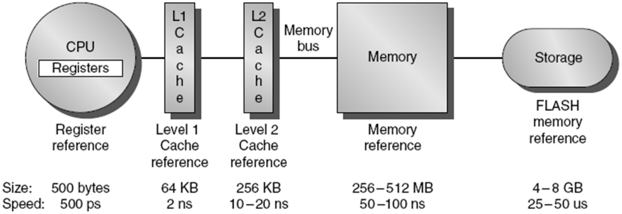


CPU: $\mathcal{O}(1\text{ns})$, L2/L3: $\mathcal{O}(10\text{ns})$, RAM: $\mathcal{O}(100\text{ns})$, disc: $\mathcal{O}(10\text{ms})$

Memory hierarchies



(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Memory hierarchies

Decreasing memory latency

- Eliminate memory operations by saving data in fast memory and reusing them, i.e., temporal locality: Access an item that was previously accessed
- Explore bandwidth by moving a chunk of data into the fast memory: spatial locality: Access data nearby previous accesses
- Overlap computation and memory access (pre-fetching; mostly figured out by compiler, but the compiler often needs help)

Memory hierarchies

Decreasing memory latency

- Eliminate memory operations by saving data in fast memory and reusing them, i.e., temporal locality: Access an item that was previously accessed
- Explore bandwidth by moving a chunk of data into the fast memory: spatial locality: Access data nearby previous accesses
- Overlap computation and memory access (pre-fetching; mostly figured out by compiler, but the compiler often needs help)

More CS terms:

- cache-hit:

Memory hierarchies

Decreasing memory latency

- Eliminate memory operations by saving data in fast memory and reusing them, i.e., temporal locality: Access an item that was previously accessed
- Explore bandwidth by moving a chunk of data into the fast memory: spatial locality: Access data nearby previous accesses
- Overlap computation and memory access (pre-fetching; mostly figured out by compiler, but the compiler often needs help)

More CS terms:

- cache-hit: required data is available in cache $\Rightarrow$ fast access

Memory hierarchies

Decreasing memory latency

- Eliminate memory operations by saving data in fast memory and reusing them, i.e., temporal locality: Access an item that was previously accessed
- Explore bandwidth by moving a chunk of data into the fast memory: spatial locality: Access data nearby previous accesses
- Overlap computation and memory access (pre-fetching; mostly figured out by compiler, but the compiler often needs help)

More CS terms:

- cache-hit: required data is available in cache $\Rightarrow$ fast access
- cache-miss:

Memory hierarchies

Decreasing memory latency

- Eliminate memory operations by saving data in fast memory and reusing them, i.e., temporal locality: Access an item that was previously accessed
- Explore bandwidth by moving a chunk of data into the fast memory: spatial locality: Access data nearby previous accesses
- Overlap computation and memory access (pre-fetching; mostly figured out by compiler, but the compiler often needs help)

More CS terms:

- cache-hit: required data is available in cache $\Rightarrow$ fast access
- cache-miss: required data is not in cache and must be loaded from main memory (RAM) $\Rightarrow$ slow access

Memory hierarchy

Simple model

1. Only consider two levels in hierarchy, fast (cache) and slow (RAM) memory
2. All data is initially in slow memory
3. Simplifications:
   - Ignore that memory access and arithmetic operations can happen at the same time
   - assume time for access to fast memory is 0
4. Computational intensity: flops per slow memory access

$$q = \frac{f}{m}, \text{ where } f \ldots \#\text{flops}, m \ldots \#\text{slow memop.}$$

Actual compute time:

$$f t_f + m t_m = f t_f \left(1 + \frac{t_m}{t_f} \frac{1}{q}\right),$$

where t_f is time per flop, and t_m the time per slow memory access.
Computational intensity should be as large as possible.

Memory hierarchy

Example: Computational intensity for Matrix-matrix multiply

- Matrix-vector multiplication: $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n, A \in \mathbb{R}^{n \times n}$

$$\boldsymbol{y} = \boldsymbol{y} + A\boldsymbol{x}$$

  flops: $\sim 2n^2$, memory read/write: $\sim 3n + n^2$
  Computational intensity: ∼2 (memory-bound!)

- Matrix-matrix multiplication: $A, B, C \in \mathbb{R}^{n \times n}$

$$C = C + AB$$

  flops: $\sim 2n^3$, memory read/write: $\sim 3n^2 + n^3$
  Computational intensity: ∼2 (memory-bound!)

Memory hierarchy

Example: Computational intensity for Matrix-matrix multiply

Can this be improved using fast memory? Yes! (Sketch—will be part of the next homework).

- Matrix-matrix multiplication with tiling: $A, B, C \in \mathbb{R}^{n\times n}$

$$C = C + AB$$

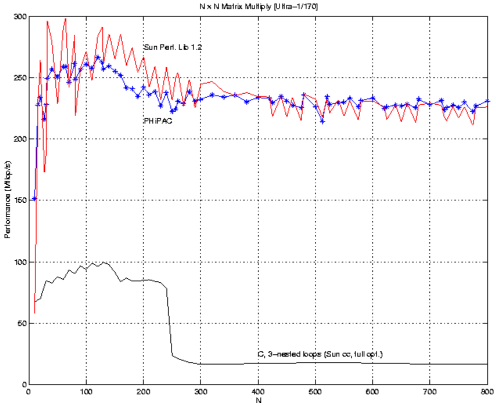
  We'll consider $N \times N$ blocks of size $b \times b$ of the matrices, i.e., $b = n/N$.

  - N^3 block reads of A, B; $2N^2$ reads/writes of C;
  - memory access: $(2N^3 + 2N^2)b^2 \sim 2Nn^2 + 2n^2$
  - Computational intensity $q \sim b$!
  - Gives a much higher computational intensity, much faster!

Memory hierarchy

Example: Matrix-matrix multiply

Comparison between naive and blocked optimized matrix-matrix multiplication for different matrix sizes.



Comparison between optimized and naive matrix-matrix multiplication on old hardware with peak of 330MFlops.
Source: J. Demmel, Berkeley

BLAS: Optimized Basic Linear Algebra Subprograms

Memory hierarchy

To summarize:

- Temporal and spatial locality is key for fast performance.
- Simple performance model: fast and slow memory; only counts loads into fast memory; computational intensity should be high.
- Since arithmetic is cheap compared to memory access, one can consider making extra flops if it reduces the memory access.
- In distributed-memory parallel computations, the memory hierarchy is extended to data stored on other processors, which is only available through communication over the network.

https://github.com/NYU-HPC19/lecture2

Outline

The module command (last week's tool)

```
module list
module avail ...
module load python/3.4
module unload texlive-2016
module whatis gcc-6.1.0
```

Valgrind and cachegrind (this week's tool)

Valgrind

- memory management tool and suite for debugging, also in parallel
- profiles heap (not stack) memory access
- simulates a CPU in software
- running code with valgrind makes it slower by factor of 10-100
- not installed by default on only available on Mac OS; use for MPI-parallel debugging on Mac limited
- Documentation: `http://valgrind.org/docs/manual/`

memcheck	cachegrind	callgrind
finds leaks inval. mem. access uninitialize mem. incorrect mem. frees	cache profiler sources of cache misses	extension to cachegrind function call graph

Valgrind and cachegrind

Usage (see examples):

Run with valgrind (no recompile necessary!)

```
valgrind --tool=memcheck [options] ./a.out [args]
```

Test examples for valgrind memcheck:

https://github.com/NYU-HPC19/lecture2

Valgrind and cachegrind

Run cachegrind profiler:

```
valgrind --tool=cachegrind [options] ./a.out [args]
```

Visualize results of cachegrind:

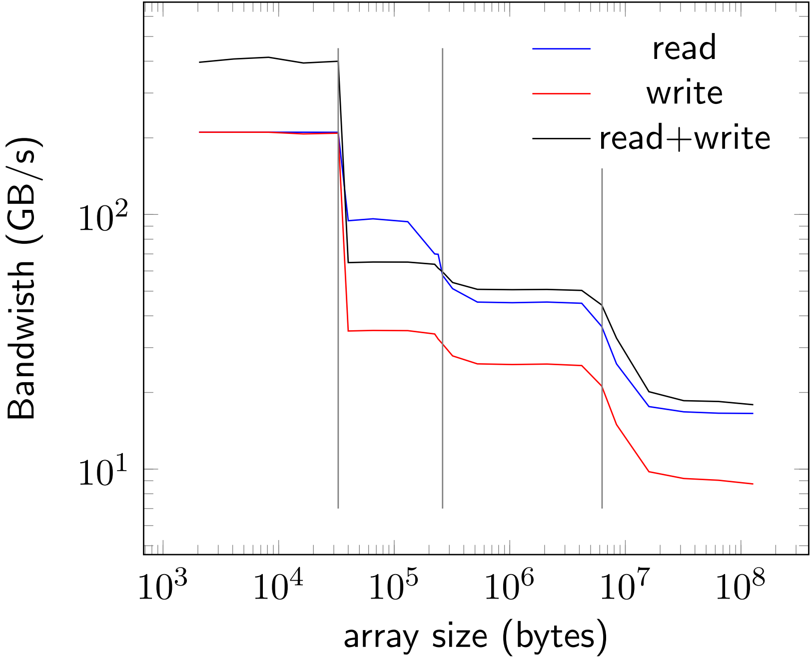
```
cg_annotate --auto=yes cachegrind.out***
```

To illustrate the use of cachegrind, we used the vector multiplication problem:

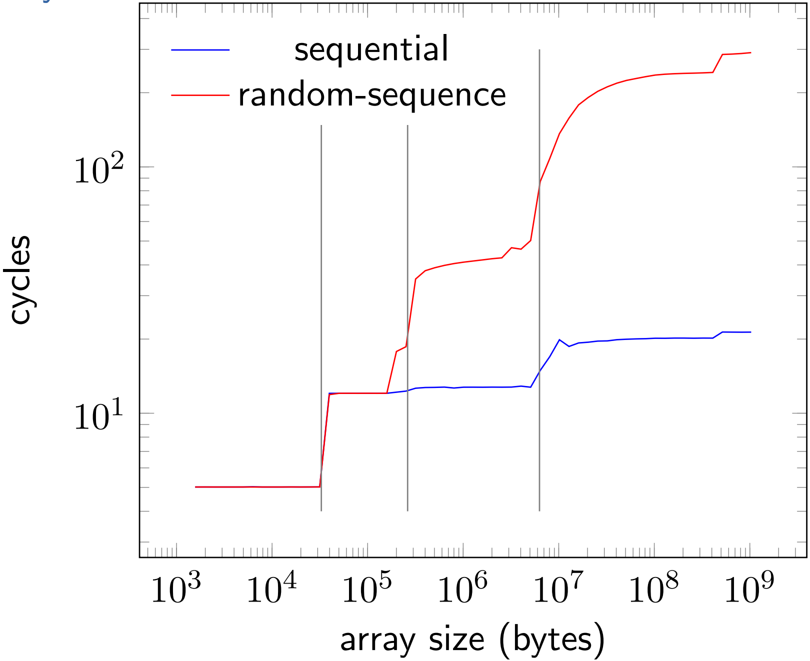
https://github.com/NYU-HPC19/lecture2

```
valgrind --tool=cachegrind
        ./inner-mem vec_size
```

Bandwidth


read
write
read+write
Bandwisth (GB/s)
10^2
10^1
10^3
10^4
10^5
10^6
10^7
10^8
array size (bytes)

Latency



sequential
random-sequence
cycles
10^2
10^1
10^3 10^4 10^5 10^6 10^7 10^8 10^9
array size (bytes)