Pitfalls for performance:
Latencies to keep in mind

Michael F. Herbst (mfh)

MRMCD 2018

8th September 2018

Contents

Contents

My field: Electronic structure theory

- Branch of theoretical chemistry
- Modelling of electrons in molecules
- Tightly related to quantum physics

$\Rightarrow$ Study linear operators $\hat{A}$ on functions Ψ ...

- ... and their approximations

Operators and functions

- An “operator”

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

- A “function”

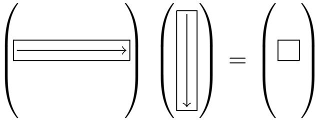
$$\underline{\boldsymbol{p}} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

- Important for understanding the physics: Find $(\alpha_i, \underline{\boldsymbol{e}}_i)$ with

$$\mathbf{A}\underline{\boldsymbol{e}}_i = \alpha_i \underline{\boldsymbol{e}}_i$$

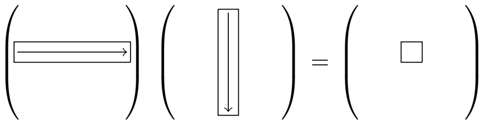
$\Rightarrow$ Diagonalisation

Matrix products



$$\mathbf{M} \quad \underline{\boldsymbol{v}} = \mathbf{M}\underline{\boldsymbol{v}}$$

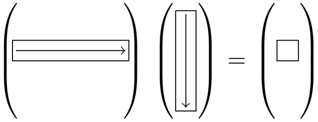
$$\sum_j M_{ij} v_j = (\mathbf{M}\underline{\boldsymbol{v}})_i$$



$$\mathbf{A} \quad \mathbf{B} = \mathbf{C}$$

$$\sum_j A_{ij} B_{jk} = (\mathbf{AB})_{ik}$$

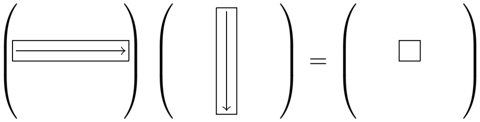
Matrix products



$$\mathbf{M} \quad \underline{\boldsymbol{v}} = \mathbf{M}\underline{\boldsymbol{v}}$$

$$\sum_j M_{ij} v_j = (\mathbf{M}\underline{\boldsymbol{v}})_i$$

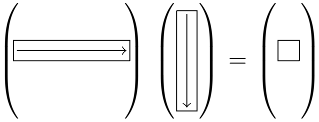
$$\mathcal{O}(n^2)$$



$$\mathbf{A} \quad \mathbf{B} = \mathbf{C}$$

$$\sum_j A_{ij} B_{jk} = (\mathbf{AB})_{ik}$$

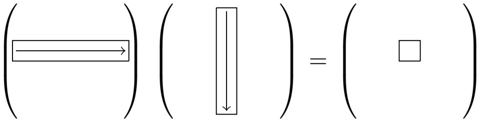
Matrix products



$$\mathbf{M} \quad \underline{\boldsymbol{v}} = \mathbf{M}\underline{\boldsymbol{v}}$$

$$\sum_j M_{ij} v_j = (\mathbf{M}\underline{\boldsymbol{v}})_i$$

$$\mathcal{O}(n^2)$$



$$\mathbf{A} \quad \mathbf{B} = \mathbf{C}$$

$$\sum_j A_{ij} B_{jk} = (\mathbf{AB})_{ik}$$

$$\mathcal{O}(n^3)$$

Diagonalisation methods

- Dense methods
- Work on the memory of $\mathbf{A}$

- Iterative methods
- Subspace-based methods
- Only diagonalise inside subspace

$\Rightarrow$ $\mathbf{A}$ can be large

$\Rightarrow$ $\mathbf{A}$ can have structure

The problem sizes of quantum chemistry

- def2-SV(P) water FCI
- Dimensionality: 43758
- Matrix elements: $2 \cdot 10^9$
- A small basis, highly accurate method

- numerical quantum-chemistry (e.g. FE)
- Dimensionality: $1 \cdot 10^6$
- Large basis

$\Rightarrow$ Storage implies hard drive

Demo: Some timings (1)

- Aim: Smallest eigenvalues of discretised Laplace operator

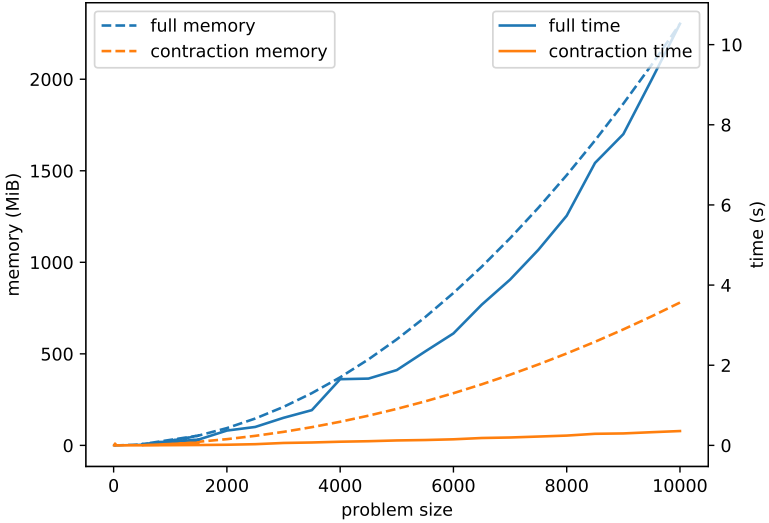
$$\mathbf{L} = \begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \end{pmatrix}$$

- Clear matrix structure

$\Rightarrow$ Putting it all in memory not advantageous

- What happens if we still do it?

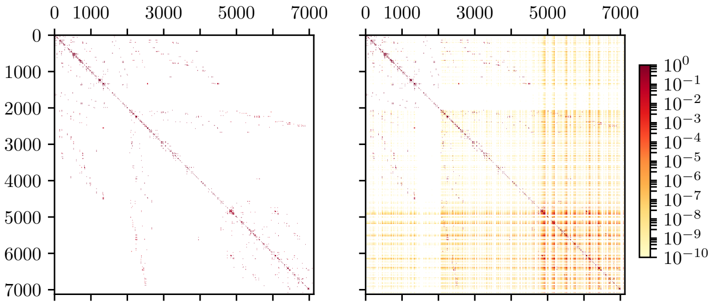
Demo: Some timings (2)



- Clearly observable speedup when avoiding memory
- Typically: Only occurs at larger problem sizes

⇒ In this case matrix structure was obvious

- Compare



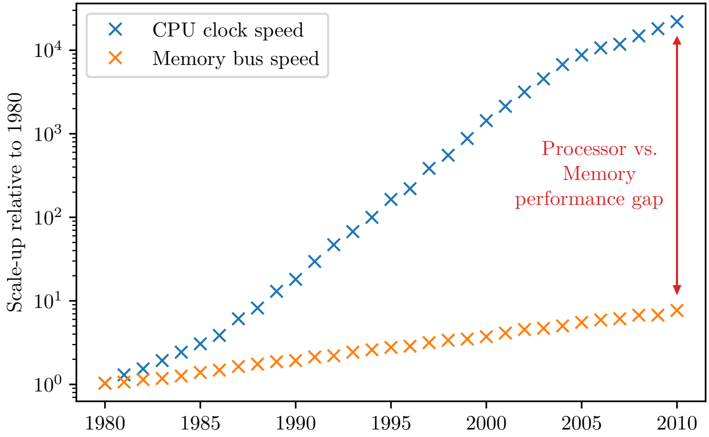
Contents

Latency numbers

Storage layer	Latency /ns	FLOPs
L1 cache	0.5	13
L2 cache	7	180
Main memory	100	2600
SSD read	$1.5 \cdot 10^4$	$4 \cdot 10^5$
HDD read	$1 \cdot 10^7$	$3 \cdot 10^8$

Data from
`https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html`
FLOPs for a Sandy Bridge 3.2GHz CPU with perfect pipelining

Hardware trends



Data from `https://dave.cheney.net/2014/06/07/five-things-that-make-go-fast`

Conclusion

- Trend is generally towards computation
- Avoiding hard disk is a clear case
- But: Trends suggest to avoid main memory as well

$\Rightarrow$ Try to design algorithms, which avoid memory

- Essentially taking disk-avoidance one step further

$\Rightarrow$ Code tends to become more complicated

Contents

Contraction-based methods

- Subspace-based algorithms only need matrix-vector product
- Only need an expression for

$$\underline{\boldsymbol{y}} = \mathbf{A}\underline{\boldsymbol{x}}$$

$\Rightarrow$ $\mathbf{A}$ not needed explicitly

- Advantages:
  - Less storage
  - Easier parallelisation
  - More freedom to exploit structure of $\mathbf{A}$
  - Structure of $\mathbf{A}$ is hidden
- Historically: Avoid hard drive

Low-rank factorisation (1)

$$\mathbf{A} = \begin{pmatrix} l_{11} & l_{12} & \cdots & \cdots & l_{1n} \\ l_{21} & l_{22} & \cdots & \cdots & l_{2n} \end{pmatrix} \begin{pmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \\ \vdots & \vdots \\ \vdots & \vdots \\ r_{n1} & r_{n2} \end{pmatrix} = \mathbf{LsR}$$

- $\mathbf{A}$ is n^2 elements
- $\mathbf{L}$ and $\mathbf{R}$ are $2n$ elements
- There is no need to build $\mathbf{A}$ fully

$\Rightarrow$ Reduction in memory

Low-rank factorisation (2)

$$\mathbf{A} = \begin{pmatrix} l_{11} & l_{12} & \cdots & \cdots & l_{1n} \\ l_{21} & l_{22} & \cdots & \cdots & l_{2n} \end{pmatrix} \begin{pmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \\ \vdots & \vdots \\ \vdots & \vdots \\ r_{n1} & r_{n2} \end{pmatrix} = \mathbf{LsR}$$

- Building $\mathbf{A}$ is $\mathcal{O}(n^2)$
- Computing $\mathbf{A}\underline{\boldsymbol{x}}$ scales as $\mathcal{O}(n^2)$

- Building $\mathbf{R}$ is $\mathcal{O}(2n)$
- Computing $\mathbf{R}\underline{\boldsymbol{x}}$ is $\mathcal{O}(2n)$

$\Rightarrow$ Reduction in computational time

Low-rank factorisation (3)

$$\mathbf{A} = \begin{pmatrix} l_{11} & l_{12} & \cdots & \cdots & l_{1n} \\ l_{21} & l_{22} & \cdots & \cdots & l_{2n} \end{pmatrix} \begin{pmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \\ \vdots & \vdots \\ \vdots & \vdots \\ r_{n1} & r_{n2} \end{pmatrix} = \mathbf{LsR}$$

- Low-rank approximation

$\Rightarrow$ $\mathbf{L}$, s, $\mathbf{R}$ might not be “physical”

$\Rightarrow$ Hard to understand

A word about low-rank in physics

- Low-rank $\approx$ redundancies / symmetries
- Explicit exploitation
  - Uses physical or numerical insight
  - For optimal performance: Maximal control of expressions required
  - Do not want to interfere with linear algebra
- Low-rank approximation:
  - Implicit exploitation
  - Leads to factorisations of matrices / tensors
  - Systematically improvable
  - Explicit symmetry sometimes hard to find or tackle

More realistic cases: Tensor contractions (1)

- Matrix:

$$\underline{\boldsymbol{y}} = f(\underline{\boldsymbol{x}} = \mathbf{M}\underline{\boldsymbol{x}}$$

- Tensor: Generalisation $\mathbf{N} = f(\mathbf{M})$
- For example

$$N_{ij} = \sum_{ab} I_{abij} M_{ab}$$

- Could depend on multiple matrices, e.g.

$$N_{ij} = \sum_{abl} I_{abij} C_{al} C_{bl}$$

- Apply this to a vector x_j, then

$$y_i = \sum_{abjl} I_{abij} C_{al} C_{bl} x_j$$

More realistic cases: Tensor contractions (2)

- Compare

  1. $$N_{ij} = \sum_{abl} I_{abij} C_{al} C_{bl}$$
  2. $$y_i = \sum_{j} N_{ij} x_j$$

  with directly

  $$y_i = \sum_{abjl} I_{abij} C_{al} C_{bl} x_j$$

- Reordering terms
- Exploiting symmetries in x_j, I_{abij}
- Exploit index selection rules
- $\mathbf{N}$ like a matrix with state $\mathbf{C}$

Combining ideas

- What if we have to combine ideas
- Things can become messy
- Different terms might have different requirements

- Expressions may become very technical
- Physically motivated modelling and interpretation difficult

- Use lazy evaluation
  - Motivated from term reordering
  - Allows to collect expression for computition
  - Optimisation not necessarily hard-coded

Lazy matrices

- Stored matrix: All elements reside in memory
- Lazy matrix:
    - Generalisation of matrices
        - Contraction expressions dressed as matrix
        - State
    - All evaluation is lazy
    - Contraction should be fast
    - For convenience: Offer matrix-like interface

  $\Rightarrow$ But: Obtaining elements expensive

Lazy matrix evaluation

- Actual expression in source code

$$\begin{aligned}\mathbf{D} &= \mathbf{A} + \mathbf{B},\\ \mathbf{E} &= \mathbf{D}\mathbf{C},\\ \underline{\boldsymbol{y}} &= \mathbf{E}\underline{\boldsymbol{x}},\end{aligned}$$

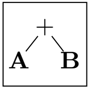
Lazy matrix evaluation

- Actual expression in source code

$$\begin{aligned} \mathbf{D} &= \mathbf{A} + \mathbf{B}, \\ \mathbf{E} &= \mathbf{D}\mathbf{C}, \\ \underline{\boldsymbol{y}} &= \mathbf{E}\underline{\boldsymbol{x}}, \end{aligned}$$

- Performed operation

$$\boxed{\mathbf{D}} = \boxed{\mathbf{A}} + \boxed{\mathbf{B}} =$$



Lazy matrix evaluation

- Actual expression in source code

$$\begin{aligned}\mathbf{D} &= \mathbf{A} + \mathbf{B},\\ \mathbf{E} &= \mathbf{DC},\\ \underline{\boldsymbol{y}} &= \mathbf{E}\underline{\boldsymbol{x}},\end{aligned}$$

- Performed operation

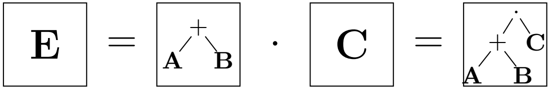
$$\boxed{\mathbf{E}} = \boxed{\mathbf{D}} \cdot \boxed{\mathbf{C}}$$

Lazy matrix evaluation

- Actual expression in source code

$$\mathbf{D} = \mathbf{A} + \mathbf{B},$$
$$\mathbf{E} = \mathbf{D}\mathbf{C},$$
$$\underline{\boldsymbol{y}} = \mathbf{E}\underline{\boldsymbol{x}},$$

- Performed operation



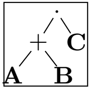
Lazy matrix evaluation

- Actual expression in source code

$$\mathbf{D} = \mathbf{A} + \mathbf{B},$$
$$\mathbf{E} = \mathbf{D}\mathbf{C},$$
$$\underline{\boldsymbol{y}} = \mathbf{E}\underline{\boldsymbol{x}},$$

- Performed operation

$$\boxed{\underline{\boldsymbol{y}}} = \boxed{\mathbf{E}}\,\boxed{\underline{\boldsymbol{x}}} =$$



$$\boxed{\underline{\boldsymbol{x}}} = (\mathbf{A} + \mathbf{B})\,\mathbf{C}\,\underline{\boldsymbol{x}}$$

Lazy matrices for contraction-based methods

- Multiple lazy matrix terms:
  - Each implements its own contraction
  - Full optimisation of contraction expressions
- Evaluation can optimise tree first
- Each term can have a physical interpretation
- Lazy matrix: Building blocks for more complicated expressions
- Transparent to end user / upper layers
  - Algorithms independent of matrix structure

Takeaway

- Memory is not generally faster than computation
- Lazy evaluation allows to exploit
  - Term reordering
  - Factorisation
  - Index selection rules
  - Streamline operations
  - Architecture-dependent optimisations
- Transparent to algorithms exploiting lazy matrices

Questions?

Paper:	`https://michael-herbst.com/molsturm-design.html`
Thesis:	`https://michael-herbst.com/phd-thesis.html`

Email:	`mfh@herbstmail.de`
Blog:	`https://michael-herbst.com/blog`