UNIVERSITÄT HEIDELBERG
ZUKUNFT
SEIT 1386

Numerically Stable Parallel Computation of (Co-)Variance

Erich Schubert, Michael Gertz

30th Int. Conf. on Scientific and Statistical Database Management (SSDBM ’18)
July 9–11, 2018, Bozen-Bolzano, Italy

Ruprecht-Karls-Universität Heidelberg
{schubert,gertz}@informatik.uni-heidelberg.de

Variance & Covariance

Variance

Variance is a widely used summary statistic:

$$\mathrm{Var}(X) = \mathrm{E}\left[(X - \mathrm{E}[X])^2\right]$$

where E[_] denotes the expected value (e.g., arithmetic average).

Variance is the "**expected squared deviation from the mean**".

Estimate the variance from a data sample ("two pass algorithm"):

1. compute $\mu_X = \frac{1}{n} \sum_i x_i$
2. compute $\mathrm{Var}(X) = \frac{1}{n-1} \sum_i (x_i - \mu_X)^2$ (or with normalization factor $\frac{1}{n}$)

From this we can, e.g., compute the standard deviation $\sigma_X := \sqrt{\mathrm{Var}(X)}$.
This is the most *common measure of spread.*

Covariance

Covariance is similar, but for two variables:

$$\mathrm{Cov}(X, Y) = \mathsf{E}\Big[(X - \mathsf{E}[X])(Y - \mathsf{E}[Y])\Big]$$

In particular, we have $\mathrm{Cov}(X, X) = \mathrm{Var}(X)$.

Used for example in:

- Pearson correlation:

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X, Y)}{\sqrt{\mathrm{Var}(X)} \cdot \sqrt{\mathrm{Var}(Y)}} = \frac{\mathrm{Cov}(X, Y)}{\sigma_X \cdot \sigma_Y}$$

- Principal Component Analysis (PCA): decomposition of the covariance matrix
- Gaussian Mixture Modeling (“EM Clustering”) uses *weighted* (co-)variance

Variance

In most statistics textbooks, we will find this "textbook algorithm":

$$\mathrm{Var}(X) = \mathrm{E}\left[(X - \mathrm{E}[X])^2\right] = \mathrm{E}[X^2] - \mathrm{E}[X]^2 \tag{1}$$

This is:

- mathematically correct (proven, c.f. König–Huygens formula, Steiner translation)
- attractive (just one pass over the data, aggregate $\sum x_i$, $\sum x_i^2$, N)

Variance

In most statistics textbooks, we will find this "textbook algorithm":

$$\mathrm{Var}(X) = \mathrm{E}\left[(X - \mathrm{E}[X])^2\right] = \mathrm{E}[X^2] - \mathrm{E}[X]^2 \tag{1}$$

This is:

- mathematically correct (proven, c.f. König–Huygens formula, Steiner translation)
- attractive (just one pass over the data, aggregate $\sum x_i$, $\sum x_i^2$, N)
- numerically problematic with *floating point computations*

➡ Use Equation (1) *only analytically*, not with floating point data.

Catastrophic Cancellation

For illustration, assume floating points with four digits of precision:

$\mathrm{E}[X^2]$		0.1234374		0.12343762
$-\ \mathrm{E}[X]^2$	-	0.0001234	-	0.12341521
$=\ \mathrm{Var}(X)$	=	0.1233140	=	0.00000000

Catastrophic Cancellation

For illustration, assume floating points with four digits of precision:

$\mathrm{E}[X^2]$		0.1234374		0.12343762
$-\ \mathrm{E}[X]^2$	-	0.0001234	-	0.12341521
$=\ \mathrm{Var}(X)$	=	0.1233140	=	0.00000000

- If $\mathrm{Var}(X) \gg \mathrm{E}[X]^2$, precision is good. But as $\mathrm{E}[X]^2 \gg 0$, we lose numerical precision.

- We can first center our data, $\mathrm{E}[X] = 0$: then $\mathrm{Var}(X) = \mathrm{E}[(X - \mathrm{E}[X])^2] \underset{\mathrm{E}[X]=0}{=} \mathrm{E}[X^2]$

  But then we need two passes over the data set. For large data, this will be 2x slower.

- Algorithms for computing variance in a *single-pass* over the data set.

E.g., Welford [Wel62], Neely [Nee66], Rodden [Rod67], Van Reeken [Van68], Youngs and Cramer [YC71], Ling [Lin74], Hanson [Han75], Cotton [Cot75], West [Wes79], Chan and Lewis [CL79], Donald Knuth in TAoCP II [Knu81], Chan et al. [CGL82; CGL83], …

Solved Problem?

Already solved since the 70s?

- Incremental (add one sample) variants of variance mostly
- Still broken (or slow) in many SQL databases & toolkits!

Let us build a small unit test with two values, $[\mu - 1, \mu + 1]$ and the mean μ:

$$\mathrm{Var}(X) = \frac{1}{2-1}\left((\mu - 1 - \mu)^2 + (\mu + 1 - \mu)^2\right) = \frac{1}{2-1}\left(-1^2 + 1^2\right) = 2$$

Easy with $\mu = 0$, but we will use $\mu = 100\,000\,000$; and thus $\mu^2 = 10^{16}$

Double precision: about 16 decimal digits (52+1 bit mantissa).

Single precision: about 6 decimal digits (23+1 bit mantissa).

➡ this breaks way too early for many use cases!

Solved Problem?

Already solved since the 70s?

- Incremental (add one sample) variants of variance mostly
- Still broken (or slow) in many SQL databases & toolkits!

PostgreSQL 9.6:

```
SELECT VAR_SAMP(x::float8), COVAR_SAMP(x,x)
FROM (SELECT 99999999 AS x UNION SELECT 100000001 AS x) AS x
```

0 ✗ 0 ✗

Solved Problem?

Already solved since the 70s?

- Incremental (add one sample) variants of variance mostly
- Still broken (or slow) in many SQL databases & toolkits!

MySQL 5.6:

```
SELECT VAR_SAMP(X)
FROM (SELECT 99999999 AS X UNION SELECT 100000001 AS X) AS X
```

2 ✓ no covariance function?

MySQL is one of the few databases that implements a numerically stable approach.

Solved Problem?

Already solved since the 70s?

- Incremental (add one sample) variants of variance mostly
- Still broken (or slow) in many SQL databases & toolkits!

MS SQL Server 2017:

```
SELECT VAR(x)
FROM (SELECT 99999999 AS x UNION SELECT 100000001 AS x) AS x;
```

0 ✗ no covariance function?

Solved Problem?

Already solved since the 70s?

- Incremental (add one sample) variants of variance mostly
- Still broken (or slow) in many SQL databases & toolkits!

HyPer 0.6:

```
SELECT VAR_SAMP(x) FROM
(SELECT 99999999::REAL AS x UNION SELECT 100000001::REAL AS x)
```

0 ✗ no covariance function?

Contributions

In this paper, we revisit the 1970s results, and contribute:

- numerically stable
- **weighted**
- **parallelizable**
- **(co-)**variance

Contributions

In this paper, we revisit the 1970s results, and contribute:

- numerically stable
- **weighted**
- **parallelizable**
- **(co-)**variance

- based on the 1970s methods
- but with arbitrary weighting
- enabling partitioned computation (AVX, ...)
- for covariance and variance

Weighted Incremental (Co-)Variance

Generalized Form

To derive the general form, we first

1. remove the scaling factor $\frac{1}{n-1}$ (resp. $\frac{1}{n}$) for now (simplification!)
2. partition the data into parts A and B
3. add weights ω_i to each observation (use $\Omega_A = \sum_{i \in A} \omega_i$)

then we get for any partition A and variables X, Y:

$$\left.\begin{aligned} \hat{x}_A &= \frac{1}{\Omega_A} \sum_{i \in A} \omega_i x_i \\ \hat{y}_A &= \frac{1}{\Omega_A} \sum_{i \in A} \omega_i y_i \end{aligned}\right\} \quad \text{weighted means}$$

$$\mathrm{Cov}(X, Y)_A \propto \quad XY_A = \sum_{i \in A} \omega_i (x_i - \hat{x}_A)(y_i - \hat{y}_A)$$

We can get the usual covariance with $\omega_i = 1$ and $\mathrm{Cov}(X, Y) = \frac{1}{\Omega - 1} XY$

Generalized Form

Using a variant of König-Huygens and some algebra (see the paper for details), we get the equations to *merge* two partitions A and B:

$$\Omega_{AB} = \Omega_A + \Omega_B$$

$$\widehat{x}_{AB} = \tfrac{1}{\Omega_{AB}}(\Omega_A \widehat{x}_A + \Omega_B \widehat{x}_B)$$

$$\widehat{y}_{AB} = \tfrac{1}{\Omega_{AB}}(\Omega_A \widehat{y}_A + \Omega_B \widehat{y}_B)$$

$$XY_{AB} = XY_A + XY_B + \tfrac{\Omega_A \Omega_B}{\Omega_{AB}} (\widehat{x}_A - \widehat{x}_B)(\widehat{y}_A - \widehat{y}_B)$$

all differences at
data precision ✓

Benefits of this form:

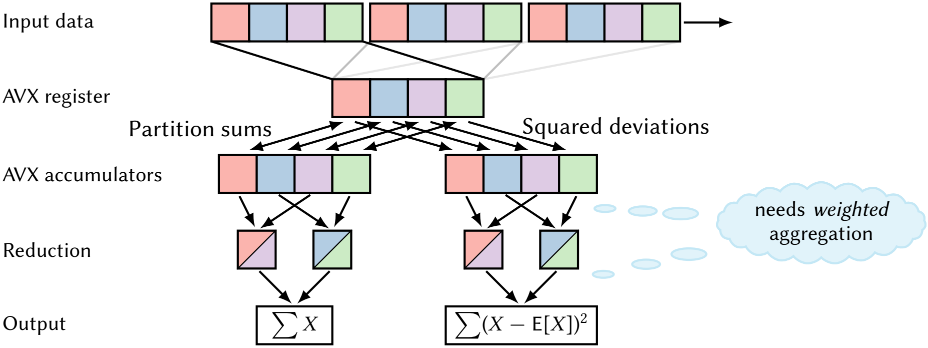
- a partition P can be summarized to a few values: Ω_P, $\widehat{x}_P$, $\widehat{y}_P$, XY_P
- two partition summaries can be combined into one
- we can partition our data using AVX, GPU, clusters, …

Note: for $|B| = 1$ and $\omega_i = 1$, this gives the "online" equations known from literature:

$$XY_{Ab} = XY_A + 0 + \tfrac{|A|}{|A|+1} (\widehat{x}_A - x_b)(\widehat{y}_A - y_b)$$

Example: AVX Parallelization of Variance

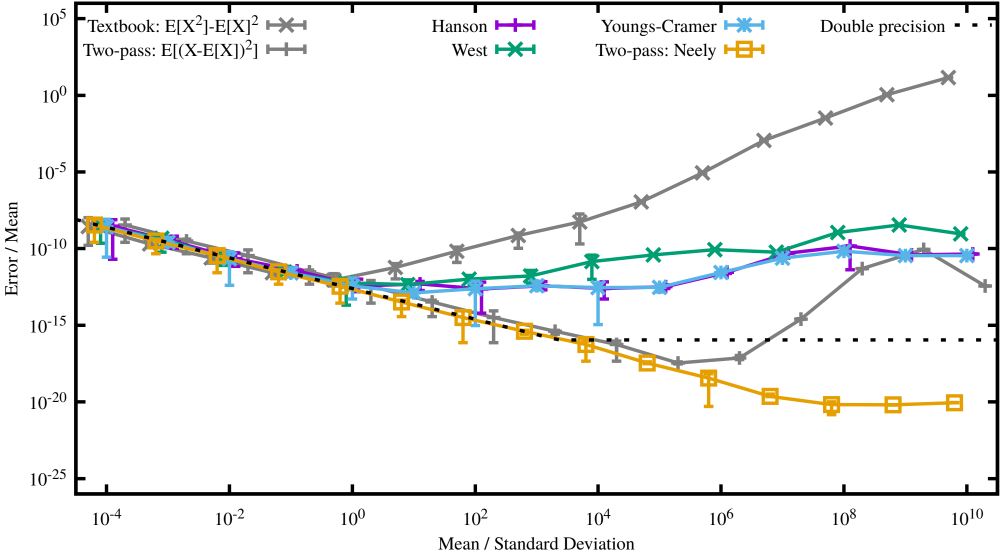
Advanced Vector Extensions (AVX) are modern vector instructions that perform the *same instruction* on 4–8 doubles (8-16 single-precision floats) in parallel.



Because the final reduction cost is negligible for larger data sets,
our parallel AVX versions are $\approx 4\times$ faster than the comparable non-parallel versions.
On GPUs, we could do this $1000\times$ parallel (but beware other GPU precision challenges)!

Experiments

Numeric precision of different (unweighted) variance algorithms:



Performance and Accuracy of Variance

Excerpt of results (see article for *many* more variants) on 100.000.000 synthetic doubles:

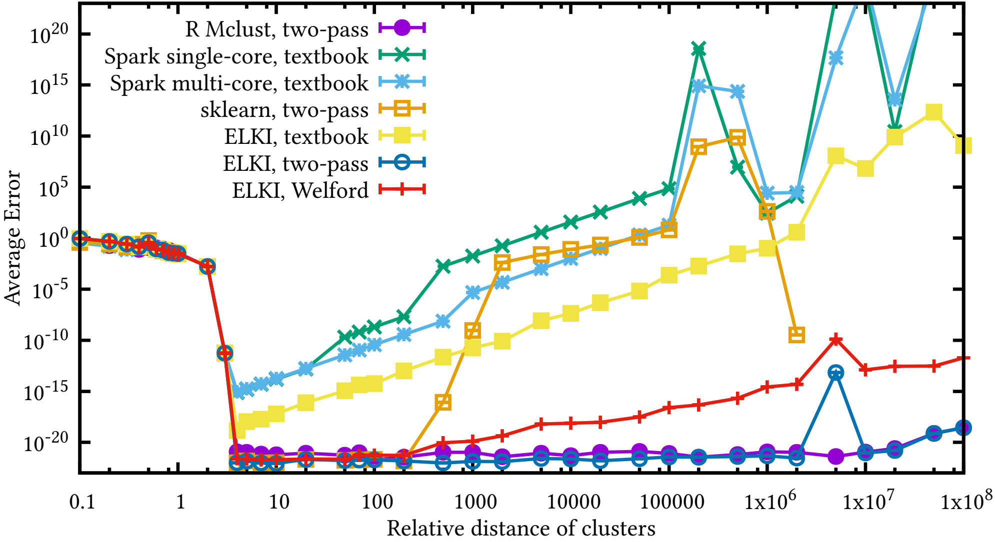
Method	Variant	Runtime (s)				Precision (decimal digits)			
Variance		Min	Mean	Median	Max	Best	Mean	Median	Worst
Textbook	double	168.85	168.97	168.93	169.22	12.848	4.086	6.150	-11.153
Welford / Knuth	double	929.17	929.97	929.93	931.18	13.224	7.441	8.787	-0.963
Youngs & Cramer	double	212.20	212.53	212.49	213.31	12.840	8.284	9.588	0.454
Welford / Knuth	AVX×4	50.93	**51.47**	**51.22**	52.83	14.041	8.892	11.306	-0.547
Youngs & Cramer	AVX×4	49.82	**52.88**	**52.98**	54.85	14.353	9.524	10.600	0.805
Two-pass	double	336.78	337.02	336.93	337.38	14.135	10.168	12.383	1.045
Two-pass	AVX×4	91.04	**92.17**	**91.74**	95.31	14.239	11.907	13.595	1.108
Two-pass (Neely)	double	337.15	337.41	337.32	338.08	14.135	12.372	12.485	10.042
Two-pass (Neely)	AVX×4	89.07	**90.47**	**89.75**	95.90	14.375	13.240	13.591	10.707

Textbook: $\mathrm{E}[X^2] - \mathrm{E}[X]^2$; Two-Pass: $\mathrm{E}[X - \mathrm{E}[X]]$; Welford: [Wel62]; Knuth: [Knu81];
Youngs-Cramer: [YC71]; Neely's two-pass improvement [Nee66]

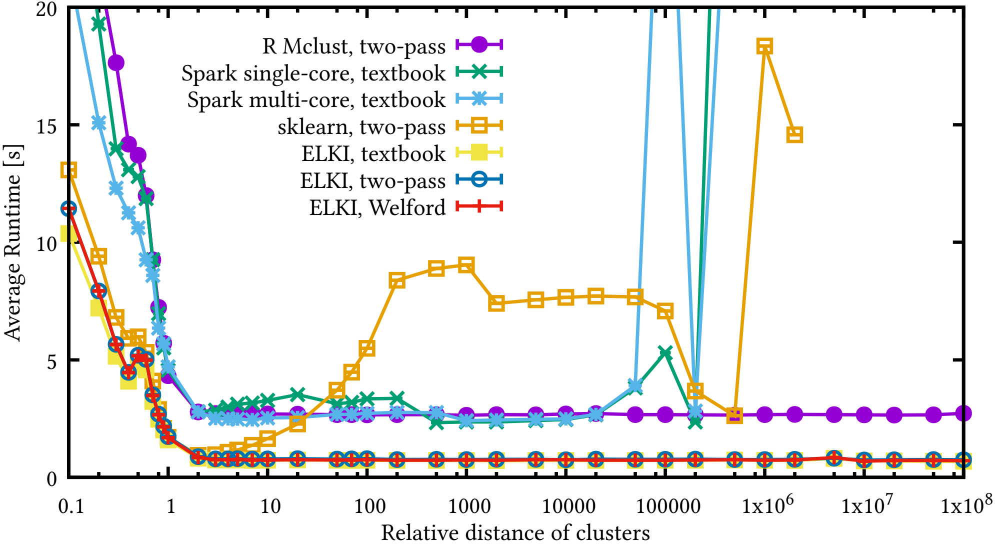
▸ Pipelining

Example: Gaussian Mixture Modeling

Comparing different EM clustering implementations:

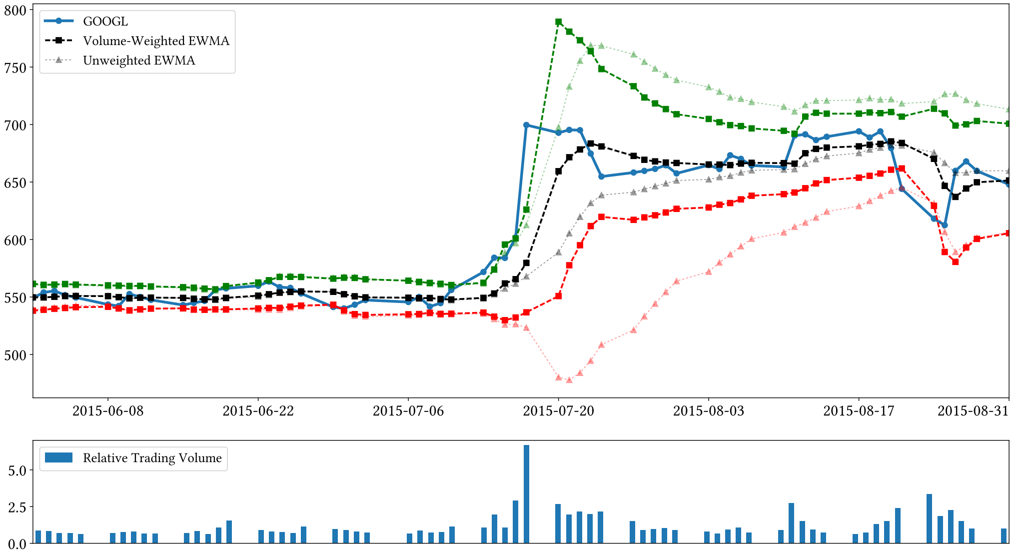


Comparing different EM clustering implementations:



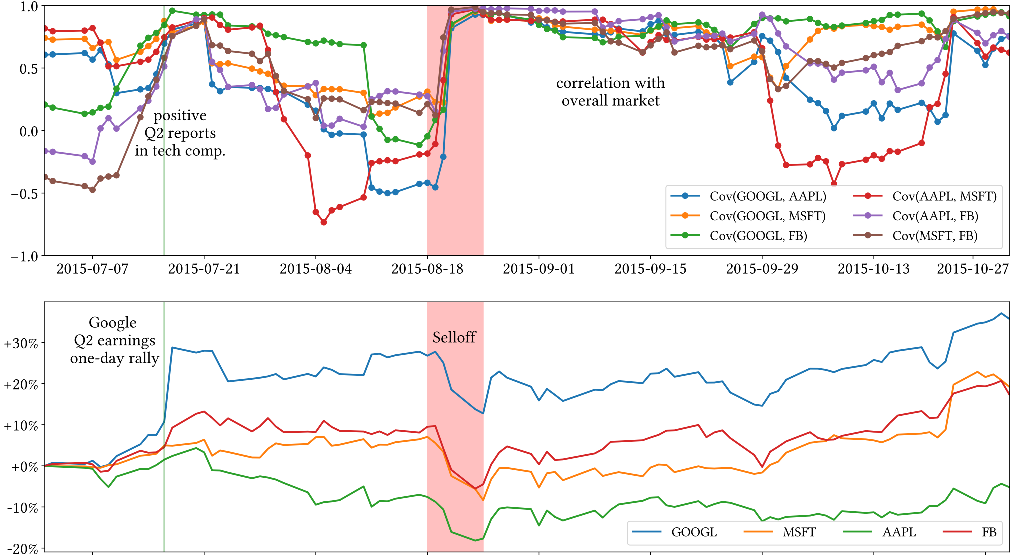
Example: Exponentially Weighted Moving Variance

Improving time series analysis with *weighted* moving standard deviation:



Example: Exponentially Weighted Moving Correlation

Weighted moving correlation of tickers (covariance normalized by standard deviation):



- **We can compute weighted (co-)variance accurately in parallel, on partitions, and distributed**
- **Numerical precision matters: do not *compute* $\mathrm{E}[X^2] - \mathrm{E}[X]^2$ with floats**
- **Even *basic* statistics can be tricky to compute reliably**
- ***Test your tools* – do not blindly trust tools**

Outline

Variance & Covariance

- Definition
- Computing (Co-)Variance
- Contributions

Weighted Incremental (Co-)Variance

- General Form
- Example: AVX Parallelization

Experiments

Bibliography

[CGL82] T. F. Chan, G. H. Golub, and R. J. LeVeque. "Updating Formulae and a Pairwise Algorithm for Computing Sample Variances". In: *COMPSTAT 1982*. 1982, pp. 30–41.

[CGL83] T. F. Chan, G. H. Golub, and R. J. LeVeque. "Algorithms for Computing the Sample Variance: Analysis and Recommendations". In: *The American Statistician* 37.3 (1983), pp. 242–247.

[CL79] T. F. Chan and J. G. Lewis. "Computing Standard Deviations: Accuracy". In: *Communications of the ACM* 22.9 (1979), pp. 526–531.

[Cot75] I. W. Cotton. "Remark on Stably Updating Mean and Standard Deviation of Data". In: *Communications of the ACM* 18.8 (1975), p. 458.

[Han75] R. J. Hanson. "Stably Updating Mean and Standard Deviation of Data". In: *Communications of the ACM* 18.1 (1975), pp. 57–58.

[Knu81] D. E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition.* Addison-Wesley, 1981. ISBN: 0-201-03822-6.

[Lin74] R. F. Ling. "Comparison of Several Algorithms for Computing Sample Means and Variances". In: *Journal of the American Statistical Association* 69.348 (1974), pp. 859–866.

[Nee66] P. M. Neely. "Comparison of Several Algorithms for Computation of Means, Standard Deviations and Correlation Coefficients". In: *Communications of the ACM* 9.7 (1966), pp. 496–499.

[Rod67] B. E. Rodden. "Error-free methods for statistical computations". In: *Communications of the ACM* 10.3 (1967), pp. 179–180.

[Van68] A. J. Van Reeken. "Letters to the editor: Dealing with Neely's algorithms". In: *Communications of the ACM* 11.3 (1968), pp. 149–150.

[Wel62] B. P. Welford. "Note on a Method for Calculating Corrected Sums of Squares and Products". In: *Technometrics* 4.3 (1962), pp. 419–420.

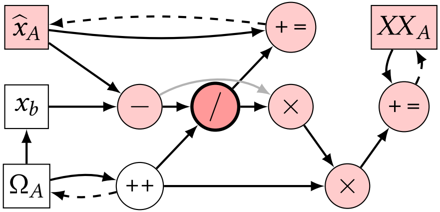
[Wes79] D. H. D. West. "Updating mean and variance estimates: an improved method". In: *Communications of the ACM* 22.9 (1979), pp. 532–535.

[YC71] E. A. Youngs and E. M. Cramer. "Some Results Relevant to Choice of Sum and Sum-of-Product Algorithms". In: *Technometrics* 13.3 (1971), pp. 657–665.

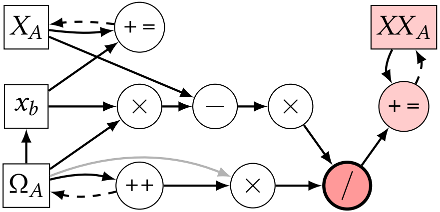
Pipelining Effects

West / Welford / Knuth: These methods use one multiplication *less*, but is slower

Youngs & Cramer: This method uses one multiplication *more*, but is faster



West [Wes79]



Youngs and Cramer [YC71]

Runtime difference can be explained by CPU pipelining (and slow divisions):

Independent of division — Depends on division result

With our AVX code, we compute the division only *once*, broadcast it,
and use it via AVX multiplication, which allows better pipelining.

◂ Pipelining