FJWNC 2025

14 March 2025

Condense & Distill:

fast distillation of large floating-point sums via condensation

Theo Mary

Sorbonne Université, CNRS, LIP6

Joint work with **Stef Graillat**

To download the paper (to appear in *SIAM J. Sci. Comput.*):

https://bit.ly/CondenseDistill

Motivation

We want to compute

$$s = \sum_{i=1}^{n} x_i$$

where

- n is large
- s is ill-conditioned:

$$\kappa = \frac{\sum_{i=1}^{n} |x_i|}{|\sum_{i=1}^{n} x_i|} \gg 1$$

Distillation

$$\sum_{i=1}^{n} x_i \xrightarrow[\textit{distillation}]{} \sum_{i=1}^{n} z_i, \qquad \text{where } \kappa(z_i) \ll \kappa(x_i)$$

Distillation

$$\sum_{i=1}^{n} x_i \xrightarrow[\textit{distillation}]{} \sum_{i=1}^{n} z_i, \qquad \text{where } \kappa(z_i) \ll \kappa(x_i)$$

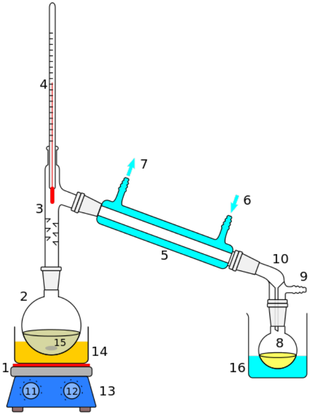
- Fast2Sum: $\mathrm{fl}(a+b) = a + b + e$, where $e \in \mathbb{F}$
- The AccSum method (Rump, Ogita, Oishi 2008): repeatedly replace (a, b) by $(\mathrm{fl}(a+b), e)$ until the sum is sufficiently well conditioned (higher $\kappa \Rightarrow$ more iterations)
- Several other variants (PrecSum, FastAccSum, FastPrecSum, ...)

Condensation

$$\sum_{i=1}^{n} x_i \xrightarrow[\textit{condensation}]{} \sum_{i=1}^{m} y_i \xrightarrow[\textit{distillation}]{} \sum_{i=1}^{m} z_i, \qquad \text{where } m \ll n \text{ and } \kappa(z_i) \ll \kappa(x_i)$$

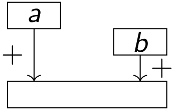
Condensation

$$\sum_{i=1}^{n} x_i \xrightarrow[\text{condensation}]{} \sum_{i=1}^{m} y_i \xrightarrow[\text{distillation}]{} \sum_{i=1}^{m} z_i, \qquad \text{where } m \ll n \text{ and } \kappa(z_i) \ll \kappa(x_i)$$

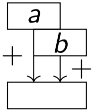


Condensation methods

Condensation methods

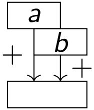


Condensation methods



Consider arithmetic with f-bit mantissa and e-bit exponent ($e = 11$ for fp64).

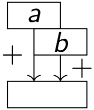
Condensation methods



Consider arithmetic with f-bit mantissa and e-bit exponent ($e = 11$ for fp64).

- One big accumulator: Kulisch method
  ... need one accumulator of
  $2^e + \log_2 n$ bits

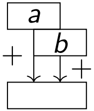
Condensation methods



Consider arithmetic with f-bit mantissa and e-bit exponent ($e = 11$ for fp64).

- One big accumulator: Kulisch method
  ... need one accumulator of
  $2^e + \log_2 n$ bits
- One accumulator per exponent:
  Malcolm method ... need 2^e
  accumulators of $f + \log_2 n$ bits

Condensation methods



Consider arithmetic with f-bit mantissa and e-bit exponent ($e = 11$ for fp64).

- One big accumulator: Kulisch method …need one accumulator of $2^e + \log_2 n$ bits
- One accumulator per exponent: Malcolm method …need 2^e accumulators of $f + \log_2 n$ bits
- Demmel–Hida: general method, balance the number and size of accumulators.

Input: n summands x_i, number of exponent bits m to extract
Output: $y = \sum_{j=1}^{2^m} A_j$

Initialize $A_j = 0$ for $j = 1, \ldots, 2^m$
for $i = 1\colon n$ **do**
  $j \leftarrow m$ leading bits of exponent(x_i)
  $A_j \leftarrow A_j + x_i$
end for

With 2^m accumulators, need F-bit mantissa with

$$F \geq f + \lceil \log_2 n \rceil + 2^{e-m} - 1$$

Distillation vs condensation

Distillation methods (AccSum, ...)

- ☺ Entirely in the working precision
- ☺ Only use standard arithmetic operations
- ☹ Strongly dependent on the conditioning
- ☹ Limited parallelism

Condensation methods (Demmel–Hida, ...)

- ☺ Independent on the conditioning
- ☺ High level of parallelism
- ☹ Require access to the exponent
- ☹ Require extended precision arithmetic

Distillation vs condensation

Distillation methods (AccSum, . . .)

- ☺ Entirely in the working precision
- ☺ Only use standard arithmetic operations
- ☹ Strongly dependent on the conditioning
- ☹ Limited parallelism

Condensation methods (Demmel–Hida, . . .)

- ☺ Independent on the conditioning
- ☺ High level of parallelism
- ☹ Require access to the exponent
- ☹ Require extended precision arithmetic

Can we avoid the use of extended precision arithmetic?

Condense & Distill, conceptually

Conceptual algorithm

$\mathbb{S} = \{x_1, \ldots, x_n\}$

Repeat for all pairs $(x_i, x_j) \in \mathbb{S}^2$ $(i \neq j)$ such that $x_i + x_j$ is exact

$\mathbb{S} \leftarrow \mathbb{S} \setminus \{x_i, x_j\}$

$\mathbb{S} \leftarrow \mathbb{S} \cup \{x_i + x_j\}$

until no such pair remains

Distill $\mathbb{S}$

- Can we easily determine when $x_i + x_j$ is exact?
- Can we bound the maximum number of leftover summands?

Outline

1. When is $x + y$ exact?
2. Condense & Distill algorithm
3. Numerical experiments

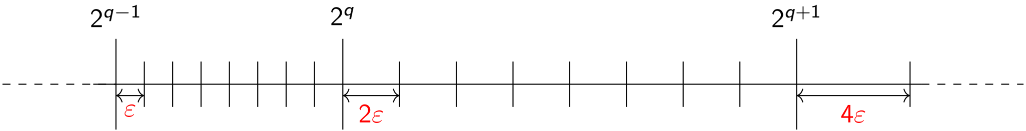
When is $x + y$ exact? Sterbenz lemma

Lemma

Let $x, y \in \mathbb{F}$. If $\frac{y}{2} \leq x \leq 2y$ then $x - y \in \mathbb{F}$, that is, $x - y$ is exact.

- Numbers of similar magnitude but of opposite sign can be added exactly.
- What about numbers of identical sign?

When is $x + y$ exact? Intuition 1



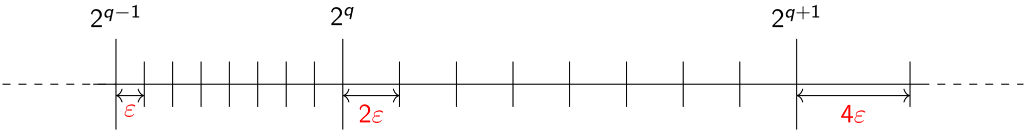
Let $x, y \in \mathbb{F} \cap [2^{q-1}, 2^q]$ such that

$$x = 2^{q-1} + k_x \varepsilon$$
$$y = 2^{q-1} + k_y \varepsilon$$

Then

$$\begin{aligned} x + y &= 2^{q-1} + k_x \varepsilon + 2^{q-1} + k_y \varepsilon \\ &= 2^q + (k_x + k_y)\varepsilon \in \mathbb{F} \text{ iff } k_x + k_y \equiv 0 \bmod 2 \end{aligned}$$

When is $x + y$ exact? Intuition 1



Similarly if

$$x = 2^{q-1} + k_x \varepsilon$$
$$y = 2^q + k_y 2\varepsilon$$

then $x + y \in \mathbb{F}$ iff

$$\begin{cases} x + y \leq 2^{q+1} \text{ and } k_x \equiv 0 \mod 2 \\ x + y > 2^{q+1} \text{ and } k_x + 2k_y \equiv 0 \mod 4 \end{cases}$$

When is $x + y$ exact? Intuition 2

$$2^q \times 101 + 2^q \times 111 = 2^q \times 1100 = 2^{q+1} \times 110.0 \in \mathbb{F}$$
$$2^q \times 101 + 2^q \times 110 = 2^q \times 1011 = 2^{q+1} \times 101.1 \notin \mathbb{F}$$

$$2^q \times 101 + 2^{q-1} \times 111 = 2^{q+1} \times 100.01 \notin \mathbb{F}$$
$$2^q \times 101 + 2^{q-1} \times 110 = 2^{q+1} \times 100.00 \in \mathbb{F}$$

When is $x + y$ exact? Theorem

Theorem

Let $x, y \in \mathbb{F}$ of the same sign $\sigma = \pm 1$ such that

$$x = \sigma(\beta^{e_x} + k_x \varepsilon_{e_x}),$$
$$y = \sigma(\beta^{e_y} + k_y \varepsilon_{e_y}).$$

Assuming (without loss of generality) that $|x| \leq |y|$, then $x + y \in \mathbb{F}$, and thus the addition is exact, iff one of the following conditions is met:

(i) $x = 0$;

(ii) $|x + y| < \beta^{e_y+1}$, $e_y - e_x \leq t - 1$, and $k_x \equiv 0 \bmod \beta^{e_y - e_x}$;

(iii) $|x + y| = \beta^{e_y+1}$, $e_y + 1 \leq e_{\max}$, $e_y - e_x \leq t - 1$, and $k_x \equiv 0 \bmod \beta^{e_y - e_x}$;

(iv) $|x + y| > \beta^{e_y+1}$, $e_y + 1 \leq e_{\max}$, $e_y - e_x \leq t - 2$, and $k_x + k_y \beta^{e_y - e_x} \equiv 0 \bmod \beta^{e_y - e_x + 1}$.

When is $x + y$ exact? Corollary

$$k_x + k_y \beta^{e_y - e_x} \equiv 0 \bmod \beta^{e_y - e_x + 1} \xrightarrow[\beta=2,\ e_x=e_y]{} k_x + k_y \equiv 0 \bmod 2$$

Corollary

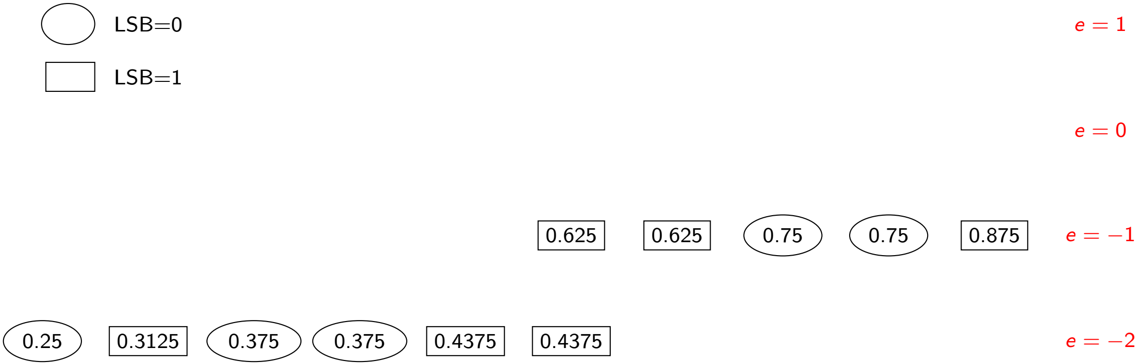
If $x, y \in \mathbb{F}$ with $\beta = 2$ have the same sign, exponent, and least significant bit, then barring overflow their addition is exact.

Consider the toy example

$$s = 0.25 + 0.3125 + 0.375 + 0.375 + 0.4375 + 0.4375 + 0.625 + 0.625 + 0.75 + 0.75 + 0.875$$

computed with 3-bit arithmetic:

$$\mathbb{F} = \{0.25, 0.3125, 0.375, 0.4375, 0.5, 0.625, 0.75, 0.875, 1, 1.25, 1.5, 1.75, 2, 2.5, 3\}$$



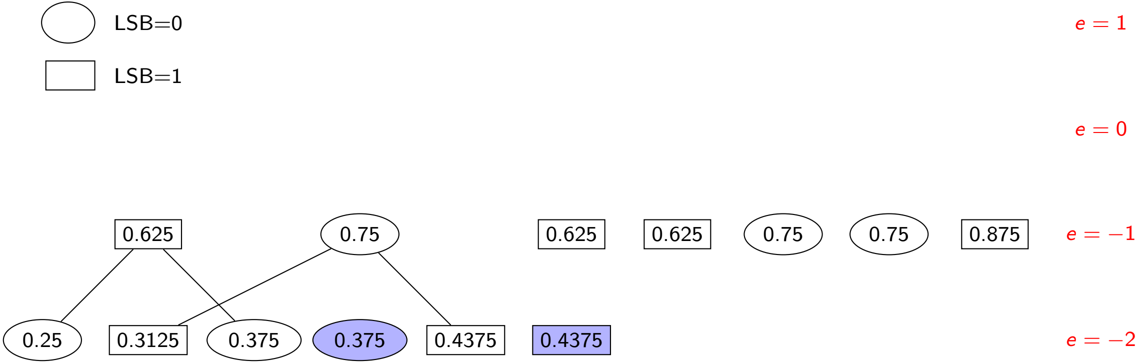
Condense & Distill (1/3)

Consider the toy example

$$s = 0.25 + 0.3125 + 0.375 + 0.375 + 0.4375 + 0.4375 + 0.625 + 0.625 + 0.75 + 0.75 + 0.875$$

computed with 3-bit arithmetic:

$$\mathbb{F} = \{0.25, 0.3125, 0.375, 0.4375, 0.5, 0.625, 0.75, 0.875, 1, 1.25, 1.5, 1.75, 2, 2.5, 3\}$$



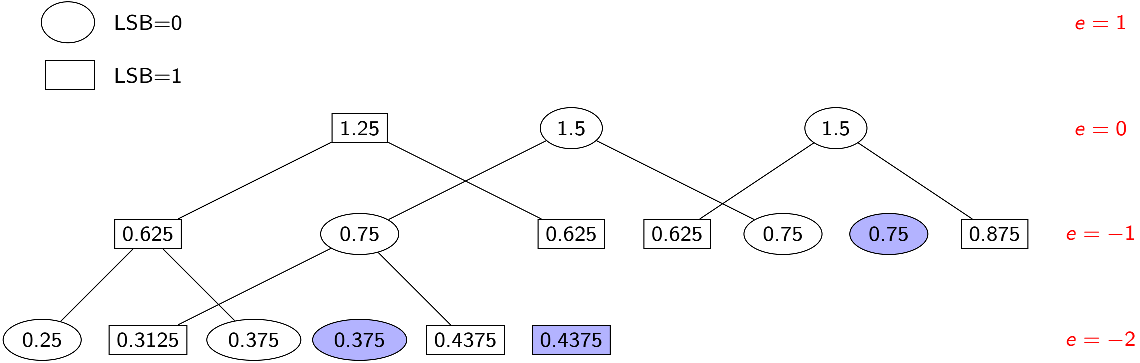
Condense & Distill (1/3)

Consider the toy example

$$s = 0.25 + 0.3125 + 0.375 + 0.375 + 0.4375 + 0.4375 + 0.625 + 0.625 + 0.75 + 0.75 + 0.875$$

computed with 3-bit arithmetic:

$$\mathbb{F} = \{0.25, 0.3125, 0.375, 0.4375, 0.5, 0.625, 0.75, 0.875, 1, 1.25, 1.5, 1.75, 2, 2.5, 3\}$$



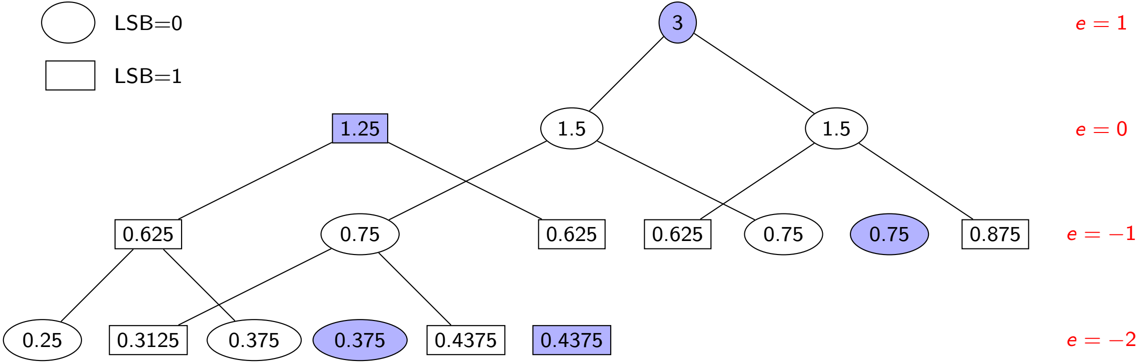
Condense & Distill (1/3)

Consider the toy example

$$s = 0.25 + 0.3125 + 0.375 + 0.375 + 0.4375 + 0.4375 + 0.625 + 0.625 + 0.75 + 0.75 + 0.875$$

computed with 3-bit arithmetic:

$$\mathbb{F} = \{0.25, 0.3125, 0.375, 0.4375, 0.5, 0.625, 0.75, 0.875, 1, 1.25, 1.5, 1.75, 2, 2.5, 3\}$$



$s = 0.375 + 0.4375 + 0.75 + 1.25 + 3$

Input: n summands x_i and a distillation method `distill`
Output: $s = \sum_{i=1}^{n} x_i$

Initialize $\mathrm{Acc}(e, s, b)$ to 0 for
$e = e_{\mathrm{min}} : e_{\mathrm{max}}$, $s \in \{-1, 1\}$, $b \in \{0, 1\}$.
for all x_i in any order **do**
  $e = \texttt{exponent}(x_i)$
  $s = \texttt{sign}(x_i)$
  $b = \texttt{LSB}(x_i)$
  $\texttt{insert}(\mathrm{Acc}, x_i, e, s, b)$
end for
$x_{\mathrm{condensed}} = \texttt{gather}(\mathrm{Acc})$
$s = \texttt{distill}(x_{\mathrm{condensed}})$

function $\texttt{insert}(\mathrm{Acc}, x, e, s, b)$
  if $\mathrm{Acc}(e, s, b) = 0$ **then**
    $\mathrm{Acc}(e, s, b) = x$
  else
    $x' = \mathrm{Acc}(e, s, b) + x$
    $\mathrm{Acc}(e, s, b) = 0$
    $b' = \texttt{LSB}(x')$
    $\texttt{insert}(\mathrm{Acc}, x', e + 1, s, b')$
  end if
end function

function $x_{\mathrm{condensed}} = \texttt{gather}(\mathrm{Acc})$
  $i = 0$
  for all nonzero $\mathrm{Acc}(e, s, b)$ **do**
    $i = i + 1$
    $x_{\mathrm{condensed}}(i) = \mathrm{Acc}(e, s, b)$
  end for
end function

Condense & Distill (3/3)

Conceptual algorithm

$\mathbb{S} = \{x_1, \dots, x_n\}$

Repeat for all pairs $(x_i, x_j) \in \mathbb{S}^2$ $(i \neq j)$ such that $x_i + x_j$ is exact

$\mathbb{S} \leftarrow \mathbb{S} \setminus \{x_i, x_j\}$

$\mathbb{S} \leftarrow \mathbb{S} \cup \{x_i + x_j\}$

until no such pair remains

Distill $\mathbb{S}$

- Can we easily determine when $x_i + x_j$ is exact? YES! It suffices to check the sign, exponent, and LSB of x_i and x_j
- Can we bound the maximum number of leftover summands? YES! At most $4L$ summands where L is the depth of the tree

$$L \leq \lceil \log_2 n \rceil + d$$

where d is independent of n and depends on the range of the values (at most 2047 in binary64)

Distillation vs condensation

Distillation methods (AccSum, . . .)

- ☺ Entirely in the working precision
- ☺ Only use standard arithmetic operations
- ☹ Strongly dependent on the conditioning
- ☹ Limited parallelism

Condensation methods (Demmel–Hida, Condense & Distill)

- ☺ Independent on the conditioning
- ☺ High level of parallelism
- ☹ Require access to the exponent + LSB
- ~~☹ Require extended precision arithmetic~~

Experimental setting

Computing environment:

- Olympe supercomputer: one node with two 18-core Intel Skylake (36 cores)
- Compiled with gfortran 9.3.0 and -O3

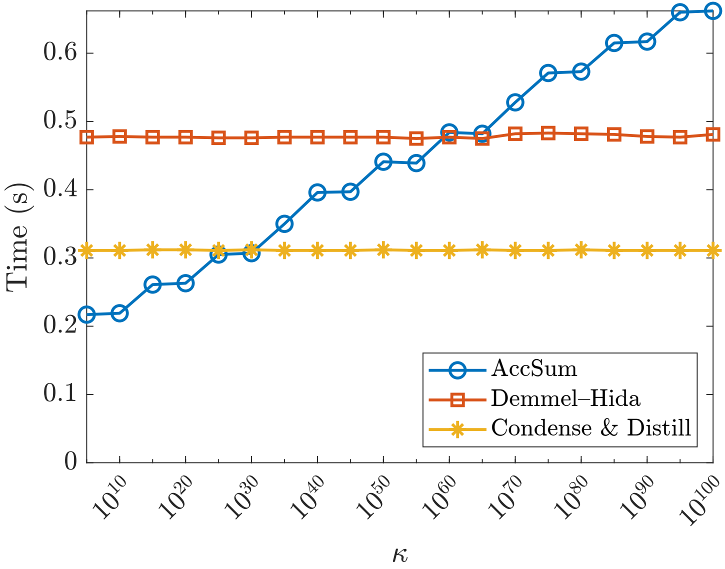
Test data:

- Generate k random summands $x_1, \dots, x_k$ in $[10^{-e}, 10^{e}]$ ($e = 32$ in the following)
- Generate another k summands $x_{k+1} = -x_1, \dots, x_{2k} = -x_k$
- Set the last summand to $x_{2k+1} = 10^{e}/\kappa$
- Randomly shuffle all summands

$\Rightarrow$ Conditioning is of order κ

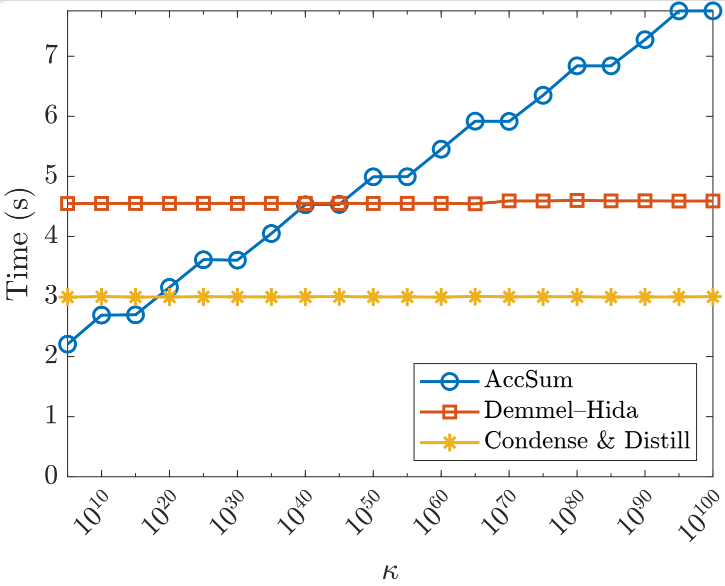
Comparison (1 core)

$n = 10^7$



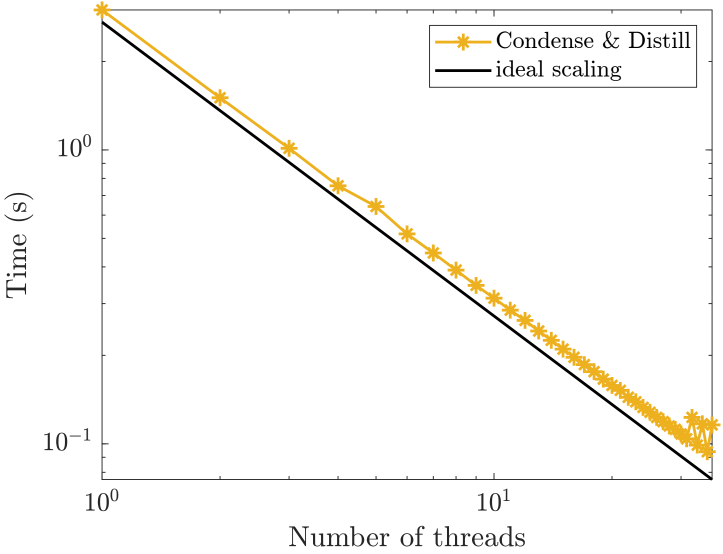
Comparison (1 core)

$n = 10^8$



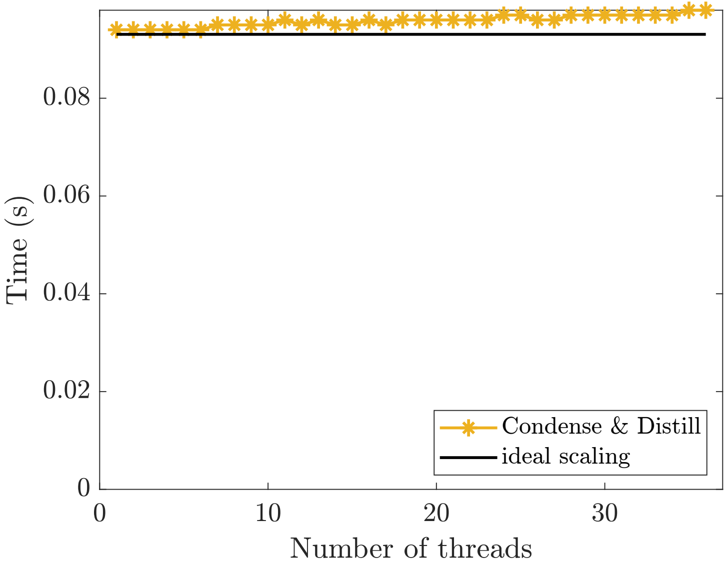
Scaling (1 $\rightarrow$ 36 cores)

Strong



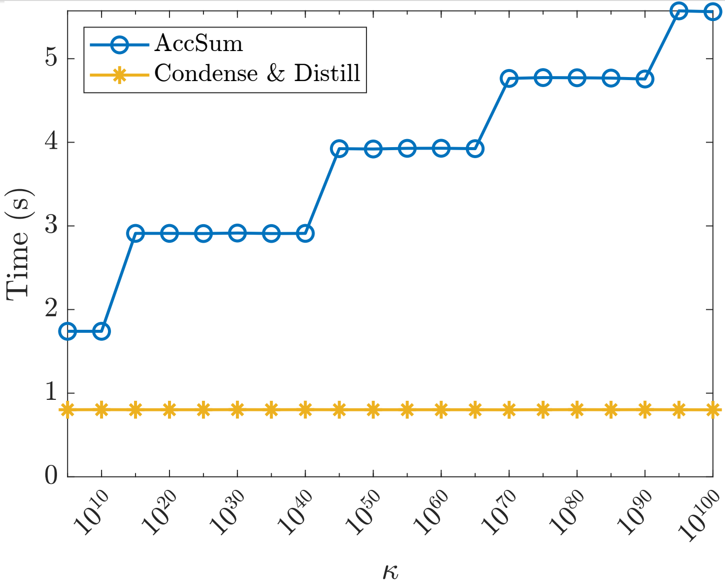
Scaling (1 $\to$ 36 cores)

Weak



Quadruple working precision (1 core)

$n = 10^7$



Conclusion

Condense & Distill:

- 35% faster than Demmel–Hida
- performance independent of conditioning κ
- entirely in the working precision
- near perfect parallel scaling

https://bit.ly/CondenseDistill

Thanks!
Questions?