Task vs. Data Parallelism: continuum

Parallelism in mathematical constructs.

First, we look at different ways to visualize/represent constructs like \((I \times A)\) and \((A \times I)\):

Consider \((\Sigma_n \times F_2)\): \((F_2 = \begin{bmatrix} 1 & -1 \end{bmatrix})\)

Matrix representation looks like:

\[
M = \begin{bmatrix}
1 & 1 & 1 & \cdots \\
1 & 1 & 1 & \cdots \\
1 & 1 & 1 & \cdots \\
\vdots & \vdots & \vdots & \ddots \\
1 & 1 & 1 & \cdots \\
\end{bmatrix}_{(K \times n)}
\]
Our transform matrix $M$ is used to transform input vector $X$ into output vector $Y$: $Y = M \cdot X$

To see how the data flows from $X$ to $Y$, we draw a **Data Flow Graph**:

For example, $(I_4 \otimes F_2)$ looks like:

\[ \begin{pmatrix} Y \\ X \end{pmatrix} \]

(data flows from right to left) \( \overset{M}{\leftarrow} \)

**Parallelization**

This visualization shows us how to partition the transform to run in parallel on multiple processors:

\[ \begin{pmatrix} Y \\ X \end{pmatrix} \]

Run on proc. $p_0$

Run on proc $p_1$

In general, $(I_n \otimes A)$ is already (naturally) parallelized for execution on up to $n$ processors.
VECTORIZATION (form of parallelism):

Example:

Goal: To vectorize \((I_2 \otimes F_2)\) on a 2-way vector machine.

Constraints: 1. Vector loads (loads to vector registers) must be done on consecutive memory elements.
   2. Loads might be expensive (cache misses etc.)
   3. Vector ops. use vertical parallelism.

Attempt #1: \(V_n\) : vector registers

\[
\begin{align*}
\text{Load} & \quad V_0 \leftarrow x_0 \\
\text{Load} & \quad V_1 \leftarrow x_1 \\
\text{Load} & \quad V_1 \leftarrow x_2 \\
\end{align*}
\]

\(\rightarrow\) might actually involve more operations (rotate, and etc.)

on most machines

Add \(V_2 \leftarrow V_0 + V_1\)

Sub \(V_3 \leftarrow V_0 - V_1\)

Store \(y_0 \leftarrow V_{2,0}\)

Store \(y_{0,1} \leftarrow V_{2,1}\)

Store \(y_2 \leftarrow V_3\)

Store \(y_3 \leftarrow V_{3,1}\)

Problem with this: Too many loads stores.

Might not get any vector speed up.

(Might get slowdown).
ATTEMPT 2

Load $V_0 \leftarrow \tau_0, x$
Load $V_1 \leftarrow x_2, x_3$

Perm $V_2 \leftarrow V_0, V_1 ((0, 2)) \} \text{ These are cheap!}$
Perm $V_3 \leftarrow V_0, V_1 ((1, 3)) \} \text{ cheap!}$

Add $V_4 \leftarrow V_2 + V_3$
Sub $V_5 \leftarrow V_2 - V_3$

Perm $V_6 \leftarrow V_4, V_5 ((0, 2)) \} \text{ cheap!}$
Perm $V_7 \leftarrow V_4, V_5 ((1, 3)) \} \text{ }$

Store $Y_0, Y_1 \leftarrow V_6$
Store $Y_2, Y_3 \leftarrow V_7$

We replaced expensive loads/stores with register permutations.

With the tensor notation:

$$(\mathbb{I}_2 \otimes F_{\tau}) = L_2^4 (F_2 \otimes I_2) L_2^4 \quad (1)$$

(Also, $A^\top (F_2 \otimes I_2) = L_2^4 (I_2 \otimes F_2) L_2^4 \quad (2)$

Helps us go from $(\mathbb{I}_n \otimes A)$ to $(A \otimes \mathbb{I}_n)$. Note that $(A \otimes \mathbb{I}_n)$ is naturally vectorized for an n-way vector machine.
In general,

\[(I_n \otimes A_m) = L^{mn}_n (A_m \otimes I_n) L^{mn}_m \quad -\text{3}\]

\[(A_m \otimes I_n) = L^{mn}_m (I_n \otimes A_m) L^{mn}_n \quad -\text{4}\]

\(\text{3} \& \text{4} \) help us convert between parallelized and vectorized forms of the same transform.