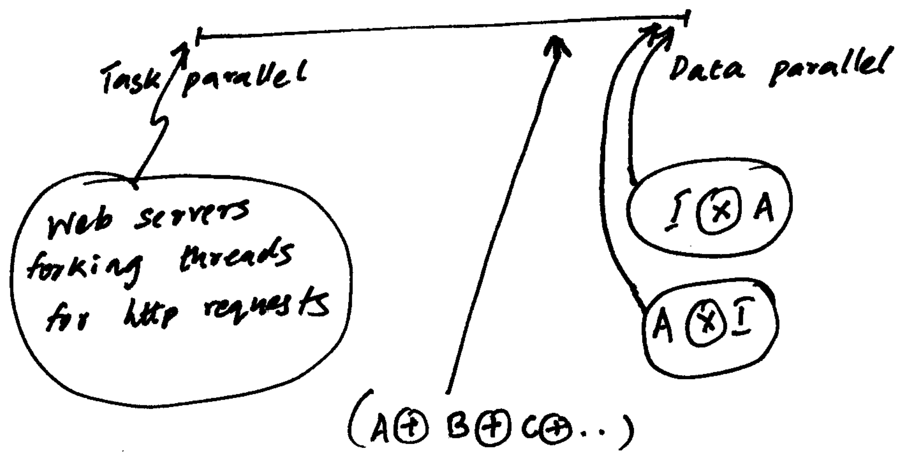
Task vs. Data Parallelism : continuum



Parallelism in mathematical constructs.

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

Consider $(I_n \otimes F_2)$: $\left(F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\right)$

Matrix representation looks like:

$$M = \begin{bmatrix} \begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix} & & & \\ & \begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix} & & \\ & & \begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix} & \\ & & & \ddots \end{bmatrix}$$

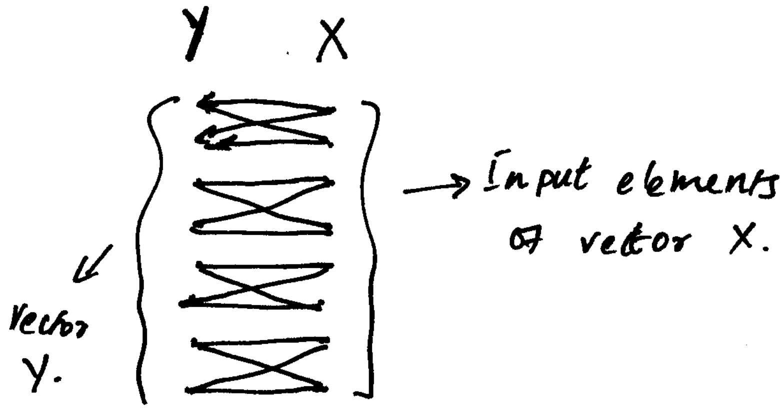
(n times)

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 <u>Data Flow Graph</u>:

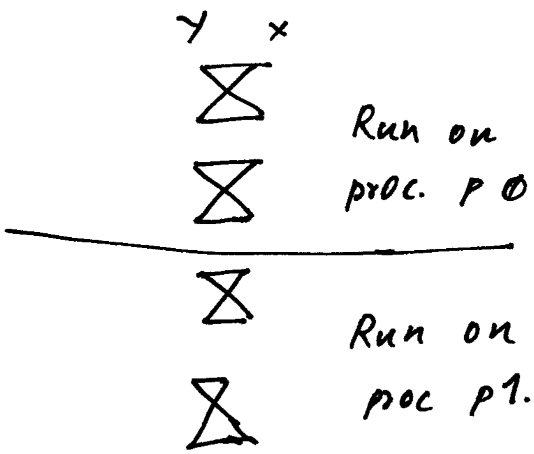
For eg., $(I_4 \otimes F_2)$ looks like:

(data flows from right to left):



<u>PARALLELIZATION</u>

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



In general, $(I_n \otimes A)$ is already (naturally) parallelized for execution on upto n processors.

VECTORIZATION (form of parallelism):

Example:

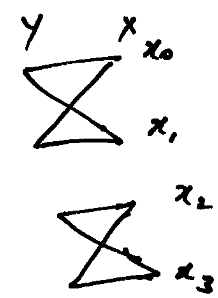
Goal: To vectorize $(I_2 \otimes F_2)$ on a 2-way vector machine.

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

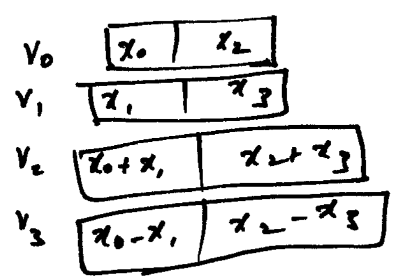


Load $V0_0 \leftarrow x_0$
Load $V0_1 \leftarrow x_2$
Load $V1_0 \leftarrow x_1$
Load $V1_1 \leftarrow x_3$

(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 $Y_0 \leftarrow V2_0$
Store $Y_1 \leftarrow V2_1$
Store $Y_2 \leftarrow V3_0$
Store $Y_3 \leftarrow V3_1$



Problem with this: Too many loads stores.

∴ Might not get any vector speed up.

(Might get slowdown).

ATTEMPT #2

Load $V_0 \longleftarrow x_0, x_1$

Load $V_1 \longleftarrow x_2, x_3$

Perm $V_2 \longleftarrow V_0, V_1 \ (0, 2)$

Perm $V_3 \longleftarrow V_0, V_1 \ (1, 3)$

These are cheap!

Add $V_4 \longleftarrow V_2 + V_3$

Sub $V_5 \longleftarrow V_2 - V_3$

Perm $V_6 \longleftarrow V_4, V_5 \ (0, 2)$

Perm $V_7 \longleftarrow V_4, V_5 \ (1, 3)$

Cheap!

Store $Y_0, Y_1 \longleftarrow V_6$

Store $Y_2, Y_3 \longleftarrow V_7$

We replaced expensive loads / stores with register permutations.

With the tensor notation: (Reg. op; Register permutations)

$$(I_2 \otimes F_2) = L^4_2 \,(F_2 \otimes I_2)\, L^4_2 \quad \text{—— ①}$$

$$\left(\text{Also, } (F_2 \otimes I_2) = L^4_2 \,(I_2 \otimes F_2)\, L^4_2\right) \quad \text{—— ②}$$

① Helps us go from $(I_n \otimes A)$ to $(A \otimes I_n)$. Note that $(A \otimes I_n)$ is naturally vectorized for an n-way vector machine.

In general,

$$(I_n \otimes A_m) = L_n^{nm} (A_m \otimes I_n) L_m^{mn} \quad (3)$$

$$(A_m \otimes I_n) = L_m^{mn} (I_n \otimes A_m) L_n^{mn} \quad (4)$$

(3) & (4) help us convert between parallelized and vetorized forms of the same transform.