Capítulo 4

HMMMR (Heterogeneous Model for Massive Multilinear Regressions)

HMMMR (Heterogeneous Model for Massive Multilinear Regressions) es un modelo que permite calcular regresiones lineales multivariables masivamente de manera eficiente haciendo uso de computacion heterogénea (procesamiento multi-core y many-core).

El objetivo principal de HMMMR es hacer una seleccion de un subconjunto de predictores que presenten mejores resultados en una regresión lineal para una determinada variable objetivo.

HMMMR como se ve en la Figura 4.1 está diseñado para ejecutarse en plataformas multi-core/many-core. Cada una de las etapas está diseñada para ejecutarse de manera eficiente en la plataforma adecuada. Adicionalmente, HMMMR cuenta con estructuras de datos diseñadas para ser procesadas en lotes con el fin de optimizar el uso de los recursos computacionales.

Como se ve en la Figura 4.1 se divide a grandes rasgos en 3 etapas de procesamiento: Multi-core I, Many-core y Multi-core II.

La etapa *multi-core I* se encarga principalmente de realizar la carga de datos desde un archivo (A), generar las combinatorias de posibles modelos usando i predictores (B), calcular el tamaño máximo de batch (C) y generar las matrices de entrada con los datos a procesar en la plataforma many-core (D). Durante esta etapa se itera desde 1 hasta el maximo de predictores mp a evaluar, generando matrices de iguales dimensiones por cada modelo a evaluar cuando se usan i predictores. La importancia

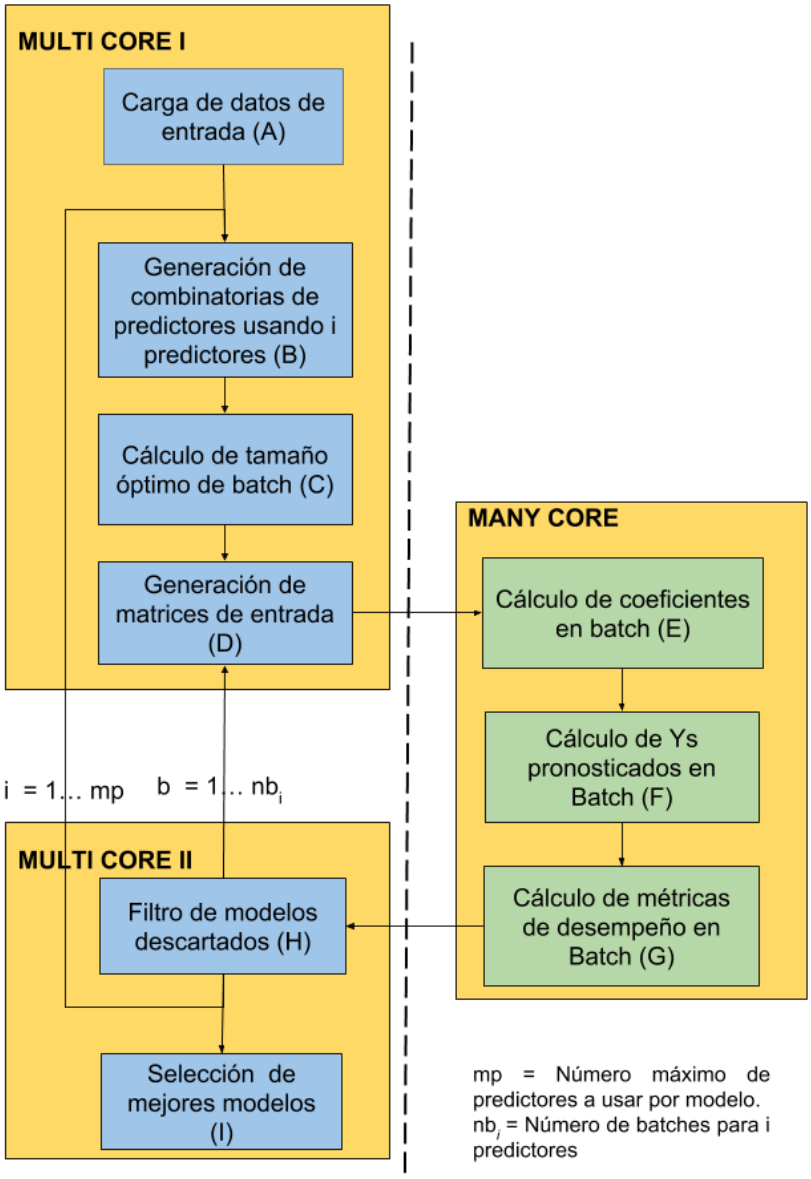

Figura 4.1: Flujo de datos HMMMR

de agrupar estos datos de entrada usando matrices de dimensiones similares radica en poder hacer uso de las capacidades de procesamiento en batch de la plataforma many-core. Cuando el numero total de regresiones a evaluar excede el tamaño máximo del batch $nb_i > 1$, esta etapa también se hace cargo de generar los datos de entrada por cada batch a procesar. Los detalles de esta etapa pueden ser vistos en la sección 4.1.

La etapa *many-core* es la que conlleva mayor esfuerzo de procesamiento ya que se encarga del calculo de las regresiones y sus metricas. Durante esta por cada batch de regresiones producido por la etapa *multi-core I* se hace inicialmente el calculo de los coeficientes de las regresiones (E) y con dichos coeficientes se calculan los valores pronosticados (F) para finalmente evaluar el desempeño de las regresiones (G). Los detalles de esta etapa pueden ser vistos en la sección 4.2.

La etapa *multi-core II* recibe las regresiones y sus metricas que fueron calculadas durante la etapa *many-core* con el fin de realizar un filtro de los modelos descartados (H) y posteriormente realizar la selección de los mejores modelos (I). Los detalles de esta etapa pueden ser vistos en la sección 4.3.

4.1. Procesamiento multi-core I

A) Carga de datos de entrada

En este proceso los datos son cargados desde un archivo en formato CSV en el cual cada columna corresponde a una de las variables (predictores y objetivo) . Este archivo debe contener *np+1* columnas donde *np* es el número de predictores. El número de filas de este archivo corresponde al número de observaciones *n*.

Las primeras *np* columnas de este archivo corresponden a los datos de los predictores y la última columna corresponde a los datos de la variable objetivo (Ver figura 4.2).

Los datos extraídos del archivo CSV son almacenados en dos estructuras de datos:

Matriz de datos de predictores

$$X = \begin{bmatrix} x_{(1,1)} & x_{(2,1)} & x_{(3,1)} & \dots & x_{(np,1)} & 1 \\ x_{(1,2)} & x_{(2,2)} & x_{(3,2)} & \dots & x_{(np,2)} & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{(1,n-1)} & x_{(2,n-1)} & x_{(3,n-1)} & \dots & x_{(np,n-1)} & 1 \\ x_{(1,n)} & x_{(2,n)} & x_{(3,n)} & \dots & x_{(np,n)} & 1 \end{bmatrix}$$

	CA	CB	CC	CD	CE	CF	CG	CH	CI	CJ	CK	CL	CM
1	21087	21087080	21087080	2108708	210870	210870	21087	210870	21087080	21087080-WL	21087080	2108708	21057060-WL_CAL_AVG
2	0	0	0	0	181.73	183.3	184.8	185.78	186.825	188.475	188.4	189.4	315.5
3	0	0	0	0	183.3	184.8	185.78	186.83	188.475	188.4	189.4	190.775	316
4	0	0	0	0	184.8	185.775	186.83	188.48	188.4	189.4	190.775	192.55	316.1
5	0	0	0	0	185.78	186.825	188.48	188.4	189.4	190.775	192.55	194.15	317
6	0	0	0	0	186.83	188.475	188.4	189.4	190.775	192.55	194.15	195.525	318.25
7	0	0	0	0	188.48	188.4	189.4	190.78	192.55	194.15	195.525	196.35	319.825
8	0	0	0	0	188.4	189.4	190.78	192.55	194.15	195.525	196.35	197.025	321.25
9	0	0	0	0	189.4	190.775	192.55	194.15	195.525	196.35	197.025	197.275	322.825
10	0	0	0	0	190.78	192.55	194.15	195.53	196.35	197.025	197.275	198.425	324.75
11	0	0	0	0	192.55	194.15	195.53	196.35	197.025	197.275	198.425	199.125	327.175
12	0	0	0	0	194.15	195.525	196.35	197.03	197.275	198.425	199.125	199.975	330.075
13	0	0	0	0	195.53	196.35	197	197.28	198.425	199.125	199.975	201.125	332.6
14	0	0	0	0	196.35	197.025	197.28	198.43	199.125	199.975	201.125	203.125	333.025
15	0	0	0	0	197.03	197.275	198.43	199.13	199.975	201.125	203.125	205.225	331.6
16	0	0	0	0	197.28	198.425	199.13	199.98	201.125	203.125	205.225	206.55	331.825
17	0	0	0	0	198.43	199.125	200	201.13	203.125	205.225	206.55	209.425	330.85
18	0	0	0	0	199.13	199.975	201.13	203.13	205.225	206.55	209.425	213.175	326.825
19	0	0	0	0	199.98	201.125	203.13	205.23	206.55	209.425	213.175	218.05	324.425
20	0	0	0	0	201.13	203.125	205.23	206.55	209.425	213.175	218.05	224.725	322.175
21	0	0	0	0	203.13	205.225	206.55	209.43	213.175	218.05	224.725	231.25	319.375
22	0	0	0	0	205.23	206.55	209.43	213.18	218.05	224.725	231.25	239.225	318.4
23	0	0	0	0	206.55	209.425	213.18	218.05	224.725	231.25	239.225	248.9	318.45
24	0	0	0	0.9	209.43	213.175	218.05	224.73	231.25	239.225	248.9	258.9	318.225
25	0	0	0.9	2.8	213.18	218.05	224.73	231.25	239.225	248.9	258.9	287.775	319.7
26	0	0.9	2.8	17.9	218.05	224.725	231.25	239.23	248.9	258.9	287.775	291.575	320.9
27	0.9	2.8	17.9	0.1	224.73	231.25	239.23	248.9	258.9	287.775	291.575	229.65	322.15
28	2.8	17.9	0.1	0	231.25	239.225	248.9	258.9	287.775	291.575	229.65	206.125	324.275
29	17.9	0.1	0	0	239.23	248.9	258.9	287.78	291.575	229.65	206.125	206.875	326.65
30	0.1	0	0	0	248.9	258.9	287.78	291.58	229.65	206.125	206.875	207.95	329.575
31	0	0	0	0	258.9	287.775	291.58	229.65	206.125	206.875	207.95	209.3	332.375
32	0	0	0	0	287.78	291.575	229.65	206.13	206.875	207.95	209.3	211.35	335.75
33	0	0	0	0	291.58	229.65	206.13	206.88	207.95	209.3	211.35	212.825	339.15
34	0	0	0	0	229.65	206.125	206.88	207.95	209.3	211.35	212.825	214.975	342.625
35	0	0	0	0	206.13	206.875	207.95	209.3	211.35	212.825	214.975	217	346.1
36	0	0	0	0	206.88	207.95	209.3	211.35	212.825	214.975	217	218.8	350.5
37	0	0	0	0	207.95	209.3	211.35	212.83	214.975	217	218.8	220.275	355.675
38	0	0	0	0	209.3	211.35	212.83	214.98	217	218.8	220.275	222.525	361.4
39	0	0	0	0	211.35	212.825	215	217	218.8	220.275	222.525	225.225	366.75
40	0	0	0	0	212.83	214.975	217	218.8	220.275	222.525	225.225	229.35	372.125
41	0	0	0	0	214.98	217	218.8	220.28	222.525	225.225	229.35	232.4	375.225
42	0	0	0	0	217	218.8	220.28	222.53	225.225	229.35	232.4	236.425	370.65
43	0	0	0	0	218.8	220.275	222.53	225.23	229.35	232.4	236.425	240.25	364.375
44	0	0	0	0	220.28	222.525	225.23	229.35	232.4	236.425	240.25	243.775	334.5
45	0	0	0	0	222.53	225.225	229.35	232.4	236.425	240.25	243.775	249.075	321.95
46	0	0	0	0	225.23	229.35	232.4	236.43	240.25	243.775	249.075	255.65	318.3
47	0	0	0	0	229.35	232.4	236.43	240.25	243.775	249.075	255.65	265.2	309.375
48	0	0	0	0	232.4	236.425	240.25	243.78	249.075	255.65	265.2	275.575	305

Figura 4.2: Ejemplo de archivo de archivo de entrada

X es una matriz compuesta de p + *1* columnas y n filas, donde p es el número total de posibles variables predictoras y n es el número de observaciones. Las primeras p columnas de la matriz contienen los datos de las variables predictoras. La ultima columna es añadida durante el proceso carga de datos y corresponde a un vector de unos que se usa de apoyo en caso de que se requiera añadir una constate al modelo.

Vector de datos de la variable objetivo

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}$$

El vector Y contiene n observaciones de la variable objetivo.

B) Generación de combinatorias de predictores

En cada iteración i de este proceso se construyen todos los posibles modelos de regresión M_i a evaluar que contengan un mismo número de predictores correspondiente a i. Este proceso se ejecuta mp veces, donde mp es el número máximo de predictores a evaluar.

$$M_i = \binom{P}{i}$$

Por cada iteración i son generados S_i posibles modelos donde:

$$S_i = \frac{|np|!}{i!(|np| - i)!}$$

Cada una de las combinaciones de predictores generadas corresponde a un posible modelo $m_{(i,n)}$ donde i es el número de predictores usado en cada iteración y n es el índice del modelo en el conjunto de modelos de i predictores.

$$m_{(i,n)} = [P_1, P_2, ..., P_i]$$

En caso de que se requiera añadir una constante al modelo, a todo modelo $m_{(i,n)}$ se le agregará la constante **1** a su conjunto de predictores.

$$m_{(i,n)} = [P_1, P_2, ..., P_i, \mathbf{1}]$$

El conjunto de modelos M_i son los modelos posibles generados durante la iteración i y estos a su vez son el insumo para la generación de las matrices de entrada.

$$M_i = \begin{bmatrix} m_{(i,1)} \\ m_{(i,2)} \\ \vdots \\ m_{(i,S_i-1)} \\ m_{(i,S_i)} \end{bmatrix}$$

C) Cálculo del tamaño máximo de batch

Para hacer un uso óptimo de las capacidades de cómputo en lote que provee la GPU es necesario determinar el número de regresiones que puede hacer al tiempo este dispositivo. Este número depende directamente de la memoria disponible en la GPU y se determina en funcion de las estructuras de datos de entrada, intermedias y de salida que estarán en la GPU durante el proceso del cálculo de los coeficientes y las métricas de las regresiones.

A continuación se listan las variables que se deben almacenar en memoria y la cantidad de números escalares que contiene cada una de ellas. En la siguiente lista n corresponde al número de observaciones y p al número de predictores que se usarán en las regresiones.

Estructuras de datos de entrada

- X: Matriz con datos de predictores. $Card(X) = (n * p)$
- X^T: Matriz transpuesta de datos de predictores.$Card(X^T) = (p * n)$
- Y: Vector de datos observados de la variable objetivo.$Card(Y) = (n)$

Estructuras de datos intermedias

- X^TX: Producto de X^T con X. $Card(X^TX) = (p * p)$.
- X^TY: Producto de X^T con Y. $Card(X^TY) = (p)$.
- $(X^TX)^{-1}$: Inversa del producto de X^T con X. $Card((X^TX)^{-1}) = (p * p)$.

Estructuras de datos de salida

- $Metric$: Valor escalar con el valor de la metrica.
- B: Coeficientes beta. $Card(B) = (p)$
- $\hat{Y}$. Vector de resultados de la regresion. $Card(\hat{Y}) = (n)$

Teniendo en cuenta las estructuras de datos a usar en el proceso, es posible calcular el número total de números escalares y los bytes en memoria requeridos por cada regresión. Cabe tener en cuenta que los posibles tipos de datos para almacenar estos valores escalares son *float32* y *float64* siendo sus tamaños 4 bytes y 8 bytes respectivamente.

$spr = Card(X) + Card(X^T) + Card(Y) + Card(X^TX) + Card(X^TY) + Card((X^TX)^{-1}) + Card(Metric) + Card(B) + Card(Y)$

$spr = np + np + n + p^2 + p + p^2 + 1 + p + n$

$spr = 2np + 2n + 2p^2 + 2p + 1$

$bpr = spr * bps$

En donde:

- bpr: Bytes requeridos por cada regresión
- spr: Numero de escalares por cada regresión
- bps: Numero de bytes por escalar.

Sabiendo de antemano la cantidad de bytes que requerirá cada regresión (bpr) y la cantidad de memoria disponible en gpu ($mgpu$) , se realiza el cálculo del tamaño de lote óptimo (mbs) aproximando el número a un múltiplo de 1000.

$$mbs = (mgpu/bps) - ((mgpu/bps) mod 1000)$$

En donde:

- $mgpu$: Memoria en bytes disponible en GPU.
- mbs: Tamaño máximo del batch.

Es posible tambien determinar el número de batches nb que se requerirá para procesar S_i regresiones para i predictores.

$$nb = S_i/mbs$$

Nota: Para las siguientes secciones de este capitulo se asume $mbs > S_i$ y $nb = 1$, es decir, el tamaño del batch máximo (mbs) es menor que el número de regresiones posibles para i predictores y por ende todas las regresiones pueden ser procesadas en un solo batch.

Cabe aclarar que la implementación del modelo si considera los casos donde $mbs < S_i$ y $nb > 1$, en dicho escenario se hace necesario repetir los procesos (D, E, F, G, H) en lotes de máximo mbs regresiones por iteración.

D) Generación de matrices de entrada

Durante este proceso se extraen los datos de los predictores para cada uno de las posibles combinatorias de modelos $m_{(i,n)}$, cada una de estas posibles combinatorias de modelos produce a su vez una matriz $X_{m_{(i,n)}}$ conteniendo los datos del modelo a evaluar.

$$X_{m_{(i,n)}} = \begin{bmatrix} x_{(P_1,1)} & .. & x_{(P_i,1)} & 1 \\ x_{(P_1,2)} & .. & x_{(P_i,2)} & 1 \\ \vdots & \ddots & .. & \vdots \\ x_{(P_1,n-1)} & .. & x_{(P_i,n-1)} & 1 \\ x_{(P_1,n)} & .. & x_{(P_i,n)} & 1 \end{bmatrix}$$

Una vez se produce cada matriz de entrada $X_{m_{(i,n)}}$ para todos los modelos en M_i, las matrices son organizadas unas sobre otras en una estructura de datos vectorial.

$$X_{M_i} = \begin{bmatrix} X_{m_{(i,1)}} \\ X_{m_{(i,2)}} \\ \vdots \\ X_{m_{(i,S_i-1)}} \\ X_{m_{(i,S_i)}} \end{bmatrix}$$

Nótese que en la matriz X_{M_i} todos los modelos contienen el mismo número de predictores. La importancia de usar el mismo número de predictores radica en poder organizar los datos de entrada de cada modelo en una estructura de datos que haga uso eficiente de memoria y que permita ser procesada en lotes.

Dado que el primer paso para realizar una regresión requiere multiplicar la matriz $X_{m_{(i,n)}}$ por su transpuesta, es necesario generar dichos datos. Para este proceso, se transponen cada una de las matrices $X_{m_{(i,n)}}$ dando lugar al vector de matrices $X^T_{M_i}$.

$$X^T_{M_i} = \begin{bmatrix} X^T_{m_{(i,1)}} \\ X^T_{m_{(i,2)}} \\ \vdots \\ X^T_{m_{(i,S_i-1)}} \\ X^T_{m_{(i,S_i)}} \end{bmatrix}$$

También se hace necesaria la generación del vector de la variable objetivo. Dado que siempre se tiene la misma variable objetivo para cada posible modelo, basta con usar un vector conteniendo S_i veces el vector de la variable objetivo.

$$Y_{M_i} = \begin{bmatrix} Y \\ Y \\ \vdots \\ Y \\ Y \end{bmatrix}$$

En este punto ya se cuenta con los datos de entrada para procesar las regresiones de manera paralela en GPU.

4.2. Procesamiento many-core

E) Cálculo de coeficientes en batch

Durante este proceso se calculan los coeficientes de regresión (β) de cada uno de las posibles combinaciones de modelos. Para calcular dichos coeficientes se hace uso de la fórmula:

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

Las operaciones que se realizan en este proceso están definidas en BLAS y LAPACK, por tanto librerías de procesamiento matemático como MKL (Math kernel library) de Intel y Cublas de Nvidia contienen implementaciones eficientes adaptadas al set de instrucciones del procesador de cada fabricante. Cabe aclarar que para aplicar el modelo propuesto en el presente trabajo, dichas librerías deben contar con soporte de procesamiento en lote (Como lo hacen la anteriormente mencionadas).

Inicialmente las matrices X_{M_i}, $X_{M_i}^T$ y Y_{M_i} son copiadas desde la memoria del host hacia la memoria de la GPU.

A continuación se realiza la multiplicación de matrices entre (X_{M_i}, $X_{M_i}^T$) y ($X_{M_i}^T$, Y_{M_i}), para esta operación se hace uso de la subrutina *GEMM* (General Matrix Multiply) del estandar BLAS en su versión en lote, esto permite realizar paralelamente esta operación.

$$X_{M_i}^T X_{M_i} = \begin{bmatrix} X_{m_{(i,1)}}^T X_{m_{(i,1)}} \\ X_{m_{(i,2)}}^T X_{m_{(i,2)}} \\ \vdots \\ X_{m_{(i,S_i-1)}}^T X_{m_{(i,S_i-1)}} \\ X_{m_{(i,S_i)}}^T X_{m_{(i,S_i)}} \end{bmatrix}$$

$$X_{M_i}^T Y_{M_i} = \begin{bmatrix} X_{m_{(i,1)}}^T Y \\ X_{m_{(i,2)}}^T Y \\ \vdots \\ X_{m_{(i,S_i-1)}}^T Y \\ X_{m_{(i,S_i)}}^T Y \end{bmatrix}$$

En la siguiente paso se realiza el cálculo de la inversa para cada uno de los productos $X_{m_{(i,j)}}^T X_{m_{(i,j)}}$ que fueron calculados en el paso anterior. Esta operación se realiza mediante factorización LU (O descomposición lower-upper) a través de las subrutinas *getrf* y *getri* defininas por LAPACK pero en su versión batched.

$$(X_{M_i}^T X_{M_i})^{-1} = \begin{bmatrix} (X_{m_{(i,1)}}^T X_{m_{(i,1)}})^{-1} \\ (X_{m_{(i,2)}}^T X_{m_{(i,2)}})^{-1} \\ \vdots \\ (X_{m_{(i,S_i-1)}}^T X_{m_{(i,S_i-1)}})^{-1} \\ (X_{m_{(i,S_i)}}^T X_{m_{(i,S_i)}})^{-1} \end{bmatrix}$$

Cabe notar que todas las matrices a las cuales se les está calculando inversa son cuadradas. Sin embargo, no todas ellas son invertibles, esto dado que la matriz puede ser singular debido a colinealidad entre los datos. Para detectar dichos casos la definición subrutinas *getrf* y *getri* de LAPACK que especifican un código de retorno en caso de que no haya sido posible calcular la inversa de la matriz, este código de retorno posteriormente será usado para hacer el filtro de modelos no válidos.

Finalmente, se realiza el producto entre $(X_{m_{(i,j)}}^T X_{m_{(i,j)}})^{-1}$ y $X_{m_{(i,j)}}^T Y$ para cada modelo $m_{(i,j)}$ con el fin de obtener los coeficientes de la regresión $B_{m_{(i,j)}}$.

$$B_{m_{(i,j)}} = (X_{m_{(i,j),}}^T X_{m_{(i,j)}})^{-1} X_{m_{(i,j)}}^T Y$$

El resultado de la anterior operación es un vector B_{M_i} de vectores $B_{m_{i,j}}$ que contiene los coeficientes para el modelo $m_{(i,j)}$.

$$B_{M_i} = \begin{bmatrix} (X^T_{m_{(i,1)}} X_{m_{(i,1)}})^{-1} X^T_{m_{(i,1)}} Y \\ (X^T_{m_{(i,2)}} X_{m_{(i,2)}})^{-1} X^T_{m_{(i,2)}} Y \\ \vdots \\ (X^T_{m_{(i,S_i-1)}} X_{m_{(i,S_i-1)}})^{-1} X^T Y \\ (X^T_{m_{(i,S_i)}} X_{m_{(i,S_i)}})^{-1} X^T_{m_{(i,S_i)}} Y \end{bmatrix} = \begin{bmatrix} B_{m_{(i,1)}} \\ B_{m_{(i,2)}} \\ \vdots \\ B_{m_{(i,S_i-1)}} \\ B_{m_{(i,S_i)}} \end{bmatrix}$$

En este punto del proceso ya se cuenta con los coeficientes beta correspondientes a cada modelo $m_{(i,j)}$.

F) Cálculo de Ys pronosticados

En este proceso se calculan los valores pronosticados a partir de los coeficientes calculados para cada uno de los modelos.

Para calcular los valores pronosticados $\hat{Y}$ basta con multiplicar la matriz conteniendo los datos de los predictores por los coeficientes del modelo.

$$\hat{Y}_{m_{(i,j)}} = X_{m_{(i,j)}} B_{m_{(i,j)}}$$

Este proceso se hace en lote para cada uno de los modelos produciendo como resultado el vector Y_{M_i} que contiene a su vez vectores con los datos calculados haciendo uso de los coeficientes.

$$\hat{Y}_{M_i} = \begin{bmatrix} X_{m_{(i,1)}} B_{m_{(i,1)}} \\ X_{m_{(i,2)}} B_{m_{(i,2)}} \\ \vdots \\ X_{m_{(i,S_{i-1})}} B_{m_{(i,S_{i-1})}} \\ X_{m_{(i,S_i)}} B_{m_{(i,S_i)}} \end{bmatrix} = \begin{bmatrix} \hat{Y}_{m_{(i,1)}} \\ \hat{Y}_{m_{(i,2)}} \\ \vdots \\ \hat{Y}_{m_{(i,S_{i-1})}} \\ \hat{Y}_{m_{(i,S_i)}} \end{bmatrix}$$

Estos valores posteriormente serán usados para el cálculo de alguna métrica de desempeño del modelo, en este caso se usará *RMSE* para evaluar el modelo.

G) Cálculo de métrica de desempeño

En este proceso se usan los valores observados Y y los valores pronosticados $\hat{Y}$ para evaluar la precisión del modelo. En el presente modelo la métrica escogida para calcular la precisión del modelo fue RMSE (Error medio cuadrático) que indica que tan cerca están los datos pronosticados a los datos reales, sin embargo es posible implementar otro tipo de métrica que se quiera evaluar.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

El primer paso para el cálculo del RMSE es el calculo del cuadrado de la diferencia de los valores observados contra los valores pronosticados. El calculo de la diferencia se hace a través de la subrutina *axpy* definida en el estandar BLAS. Para el cálculo de los cuadrados se implementó un kernel que calcula el cuadrado elemento a elemento. Cabe notar que en este caso no es necesario que la operación se haga de manera batch, ya que solo hay una dimensión de datos y la operación se hará elemento a elemento.

$$\hat{Y} - Y = \begin{bmatrix} \begin{bmatrix} (\hat{y}_{m_{(i,1)}1} - y_1)^2 \\ (\hat{y}_{m_{(i,1)}2} - y_2)^2 \\ (\hat{y}_{m_{(i,1)}3} - y_3)^2 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} (\hat{y}_{m_{(i,S_i)}1} - y_1)^2 \\ (\hat{y}_{m_{(i,S_i)}2} - y_2)^2 \\ (\hat{y}_{m_{(i,S_i)}3} - y_3)^2 \end{bmatrix} \end{bmatrix}$$

Una vez se cuenta con la diferencia de cuadrados, es necesaria hacer la sumatoria de las diferencias para cada modelo. Dado que las diferencias al cuadrado de todos los modelos se encuentran en el mismo vector, es necesario calcular cada una de las sumatorias independientemente. Este problema de la sumatoria, puede ser solucionado en lote mediante la multiplicación de cada vector por un vector de unos de igual cantidad de elementos.

$$\begin{bmatrix} \sum_{n=1}^{t}(\hat{Y}_{m_{(i,1)}n,} - Y_n) \\ \vdots \\ \sum_{n=1}^{t}(\hat{Y}_{m_{(i,S_i)}n,} - Y_n) \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} (\hat{y}_{m_{(i,1)}1} - y_1)^2 \\ (\hat{y}_{m_{(i,1)}2} - y_2)^2 \\ (\hat{y}_{m_{(i,1)}3} - y_3)^2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} (\hat{y}_{m_{(i,S_i)}1} - y_1)^2 \\ (\hat{y}_{m_{(i,S_i)}2} - y_2)^2 \\ (\hat{y}_{m_{(i,S_i)}3} - y_3)^2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \end{bmatrix}$$

Dando como resultado la sumatoria de los cuadrados de las diferencias en cada una de las regresiones como un valor escalar.

$$\begin{bmatrix} \begin{bmatrix} (\hat{y}_{m_{(i,1)}1} - y_1)^2 \\ (\hat{y}_{m_{(i,1)}2} - y_2)^2 \\ (\hat{y}_{m_{(i,1)}3} - y_3)^2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} (\hat{y}_{m_{(i,S_i)}1} - y_1)^2 \\ (\hat{y}_{m_{(i,S_i)}2} - y_2)^2 \\ (\hat{y}_{m_{(i,S_i)}3} - y_3)^2 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \end{bmatrix} =$$

$$\begin{bmatrix} (\hat{y}_{m_{(i,1)}1} - y_1)^2 + (\hat{y}_{m_{(i,1)}2} - y_2)^2 + (\hat{y}_{m_{(i,1)}3} - y_3)^2 \\ \vdots \\ (\hat{y}_{m_{(i,S_i)}1} - y_1)^2 + (\hat{y}_{m_{(i,S_i)}2} - y_2)^2 + (\hat{y}_{m_{(i,S_i)}3} - y_4)^2 \end{bmatrix}$$

Con el valor de las sumatorias calculado, se dividen cada una de las sumatorias en n (número de datos observados ó pronosticados). Esta división se hace a través de la multiplicación escalar de cada valor del vector por $1/n$. Finalmente se calcula la raíz cuadrada de cada elemento. Ninguna de esta operaciones es necesaria hacerla en lote ya que se calcula elemento a elemento.

$$\begin{bmatrix} RMSE_{m_{(i,1)}} \\ \vdots \\ RMSE_{m_{(i,S_i)}} \end{bmatrix} = \begin{bmatrix} \sqrt{\frac{\sum_{i=1}^{n}(\hat{Y}_{m_{(i,1)}n,} - Y_n)}{n}} \\ \vdots \\ \sqrt{\frac{\sum_{i=1}^{n}(\hat{Y}_{m_{(i,s_i)}n,} - Y_n)}{n}}) \end{bmatrix}$$

4.3. Procesamiento multi-core II

H) Filtro de modelos descartados

En este proceso se copian los resultados de las regresiones de la memoria de la GPU a la memoria del host. Dichos resultados incluyen: valores observados, valores simulados, coeficientes, métricas de desempeño y los resultados de las operaciones de inversa. Estos últimos, indican si fue posible o no realizar la operación de inversa de una matriz para un modelo determinados, si la operación no se pudo realizar de manera correcta el código de retorno es 0.

Haciendo uso del retorno de dicha función es posible descartar por completo aquellos posibles modelos en los cuales no fue posible realizar la regresión.

I) Selección de mejores modelos

Durante este proceso se usan los resultados de las regresiones (especificamente el RMSE) con el fin de escoger el conjunto de predictores que mejores resultados presentan para las regresiones multivariables.

Cabe aclarar que dichos resultados carecen de diágnostico y por ende posterior a este proceso se recomienda hacer un filtro de los modelos seleccionados a través de otros análisis estadísticos que permitan diagnosticar la regresión (Multicolinealidad, no linealidad, significancia estadística de las variables, entre otros). Sin embargo, nótese que este análisis estadístico solo debe ser ejecutado sobre un sub conjunto de combinatorias de predictores de menor tamaño que todas las posibles combinatorias de predictores.

4.4. Implementación

HMMMR es un modelo agnóstico a la tecnología, es decir, puede implementarse sobre cualquier arquitectura CPU/GPU. En este caso, HMMMR fue implementado sobre tecnologías Nvidia®. ya que esta tecnología cuenta con una mayor participación

(66.3 %) vs su competidor AMD (33.7 %) en el mercado de graficas discretos Teske (2018).

Para la implementación del modelo se seleccionaron las siguientes tecnologías.

- *Python*: Lenguaje de programación libre y multipropósito.
- *Numpy*: Biblioteca informática de python que cuenta con estructuras de datos que soportan arrays y matrices multidimensionales. Esta biblioteca usa tipos de datos que permiten gestionar la memoria de una manera eficiente.
- *Pycuda*: Libreria que permite acceder a el API de computo paralelo de CUDA.
- *Cublas*: Libreria que implementa subrutinas definidas en el estandar BLAS y LAPACK. Algunas de estas subrutinas cuentan con versiones de procesamiento en batch. Las subrutinas usadas de esta libreria fueron:
  - *cublasSaxpy/cublasDaxpy* Implementaciones en lote de la subrutina *axpy* del estandar BLAS. Se utiliza para calcular suma de vectores. *cublasSaxpy* y *cublasDaxpy* son usados para números flotantes de 32 y 64 bits respectivamente.
  - *cublasSgemmBatched/cublasDgemmBatched*: Implementaciónes en lote de la rutina GEMM de BLAS para la multiplicación de matrices. *cublasSgemmBatched* y *cublasDgemmBatched* son usados para números flotantes de 32 y 64 bits respectivamente.
  - *cublasSgetrfBatched/cublasDgetrfBatched* Implementaciones en lote de la subrutina *getrf* del estandar LAPACK. Calcula la factorización LU de una matriz. *cublasSgetrfBatched* y *cublasDgetrfBatched* son usados para números flotantes de 32 y 64 bits respectivamente.
  - *cublasSgetriBatched/cublasSgetriBatched* Implementaciones en lote de la subrutina *getri* del estandar LAPACK. Calcula la inversa de una matriz LU factorizada. *cublasSgetriBatched* y *cublasSgetriBatched* son usados para números flotantes de 32 y 64 bits respectivamente.

La implementación de HMMMR está disponible en el siguiente repositorio Repositorio en GitHub de HMMMR.