Resolución de Sistemas de Ecuaciones Lineales Enormes de Rango Deficiente

Gregorio Quintana ©

Departamento de Ingeniería y Ciencia de Computadores
Universidad Jaume I

MINISTERIO DE CIENCIA E INNOVACIÓN
Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
AGENCIA ESTATAL DE INVESTIGACIÓN

Contenido

Introducción

Problema

- Normalmente, se desea resolver uno de los dos problemas siguientes:
  - $Ax = b$, donde x y b son vectores.
  - $AX = B$, donde X y B son matrices.
- El segundo problema consiste en múltiples problemas del primer tipo con la misma matriz de coeficientes.

Motivación

- Algunos problemas de las Ciencias e Ingenierías requieren la resolución de sistemas de ecuaciones lineales tan grandes que no caben en la memoria central de un ordenador.

  1. Detección de bancos de peces en los fondos marinos.
  2. Algunos fondos de inversión (usando miles de nodos de AWS) resuelven sistemas de ecuaciones en los que A tiene unas dimensiones de alrededor de $1\,000\,000 \times 1\,000\,000$.

     Sólo el almacenamiento de A requiere 8 TB.
  3. La generación de imágenes médicas TAC con resolución 512×512 requiere la resolución de sistemas de ecuaciones de dimensión $266\,500 \times 262\,144$.

     Sólo el almacenamiento de A requiere 560 GB.
  4. etc.

Motivación (Cont.)

- El precio de la memoria RAM es bastante alto.
- Si se desea almacenar todos los datos en RAM, el precio total puede ser prohibitivo.
- Además, cuanta más RAM tiene un ordenador, más suele aumentar el coste del resto del ordenador.
- ¿Qué se puede hacer para resolver sistemas de ecuaciones enormes a un coste bajo?

Intensidad Aritmética

- Velocidad actual del *hardware*:
  1. Una CPU puede realizar varios *flops* en un ciclo de reloj.
  2. Un acceso a memoria central cuesta unos 100 ciclos.
  3. Un acceso al disco duro cuesta muchísimo más.
- Intensidad Aritmética: División o ratio entre el número de *flops* y el número de *memops* (accesos a memoria).
- Ejemplo:

```
1  for( i = 0; i < m; i++ ) {
2    sum = 0.0;
3    for( j = 0; j < n; j++ ) {
4      sum += a[ i ][ j ] * x[ j ];
5    }
6    y[ i ] = sum;
7  }
```

- Concepto clave en Computación de Altas Prestaciones.

Sistemas de Tamaño Moderado de Rango Completo

Resolución de Sistemas Moderados de Rango Completo

- Si la matriz de coeficientes **tiene rango completo**, la resolución suele basarse en las factorizaciones lineales de Cholesky, LU o QR.
- La factorización QR es el doble de costosa que la LU, pero es más estable numéricamente.
- La secuencia de pasos para resolver el sistema $Ax = b$ con la factorización QR es la siguiente:

  1. Obtención de Q y R tal que: $A = Q\,R$
  2. Aplicación de Q^T a b: $y = Q^T b$
  3. Resolución del sistema triangular superior: $R\,x = y$

- Si los datos caben en RAM, existen implementaciones muy eficientes de dichas operaciones, como por ejemplo en la biblioteca LAPACK.

Sistemas de Tamaño Moderado de Rango Deficiente

Resolución de Sistemas Moderados de Rango Deficiente

- Si la matriz de coeficientes **tiene rango deficiente**, el sistema puede no tener solución o tener múltiples.
  En tal caso, se busca calcular lo siguiente (problema lineal de cuadrados mínimos): $\text{mín}_x \, ||Ax - b||_2$.
- La resolución de este problema puede basarse en las siguientes herramientas:
  1. Factorización QR con pivot. de columnas: $AP = QR$
  2. Factorización SVD: $A = U\Sigma V^T$

     Mayor coste computacional que la anterior.
- Ambas realizan muchas operaciones matriz-vector, con baja intensidad aritmética. Por tanto, bajo rendimiento.
- Si los datos caben en RAM, existen implementaciones de dichas operaciones, como por ejemplo en la biblioteca LAPACK.

Sistemas de Tamaño Enorme de Rango Completo

Resolución de Sistemas ENORMES de Rango Completo

- Si los datos NO caben en RAM, habrá que guardarlos en disco. Y los discos son bastante lentos.
- En dos trabajos ya publicados con Mónica Chillarón y Vicente Vidal conseguimos resolver sistemas enormes almacenados en disco con una velocidad similar a la obtenida cuando se trabaja en RAM.
- Estos trabajos fueron aplicados a la generación de imágenes TAC.
- A continuación se describen las técnicas empleadas.

Procesamiento y Particionado de Matrices Enormes en Bloques

- Las matrices deben ser particionadas en bloques.
- La secuencia habitual para ejecutar cada tarea es la siguiente:
  1. Se traen los bloques necesarios desde disco a memoria central.
  2. Se realiza el procesamiento de dichos bloques en memoria central.
  3. Se llevan los bloques *modificados* de vuelta desde memoria central a disco.
- Este tipo de técnicas se llaman OOC (*Out-Of-Core*).

Geometría y Dimensiones de los Bloques

- ¿Cuáles son la mejor geometría y las mejores dimensiones de los bloques?

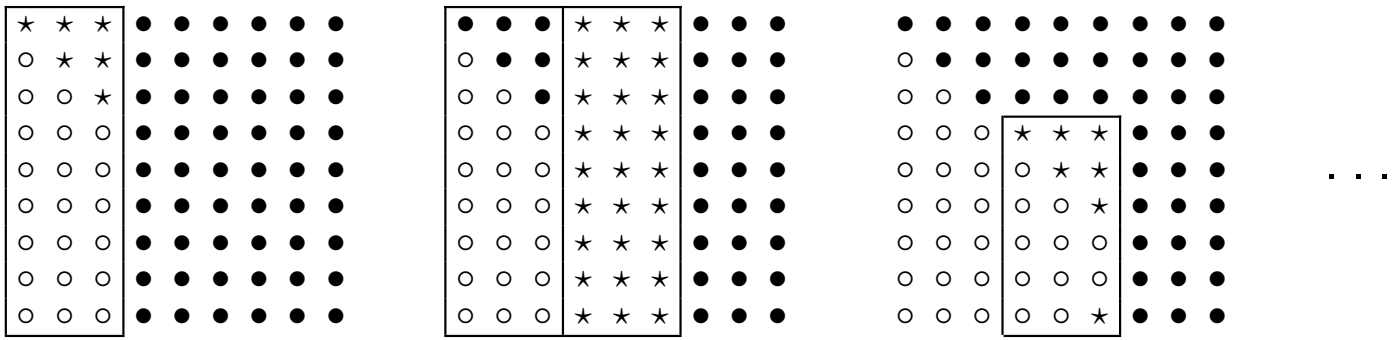
  1. Si se trabaja con bloques fila o bloques columna, el tamaño total en octetos depende directamente de las dimensiones de la matriz (m y n).

     Eso hace que el tamaño no pueda ser ajustado de forma óptima.

  2. Si se trabaja con bloques cuadrados (excepto en los bordes si el tamaño de bloque no es divisor de m y n), se puede ajustar un tamaño de bloque más óptimo.

Factorización QR *blocked* Tradicional

- Funciona como el algoritmo tradicional, pero en cada iteración se procesan varias columnas.



(1) After Compute_dense_QR(A_0)

(2) After Apply_left_Qt_of_Dense_QR(A_0, A_1)

(3) After Compute_dense_QR(A_1)

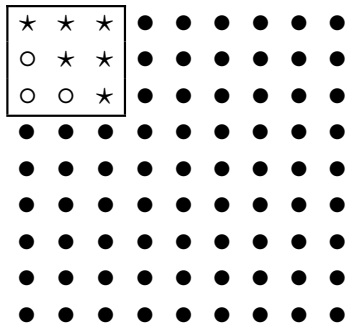
- Con un particionado por columnas, el tamaño del bloque procesado varía mucho.

Factorización QR Incremental o Factorización QR *algorithm-by-blocks*

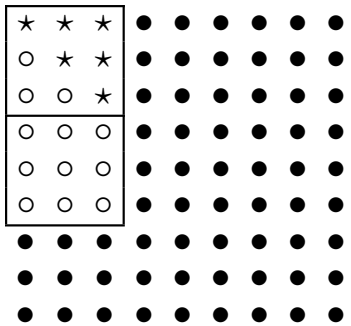
- Si se quiere trabajar con bloques cuadrados, entonces hay que emplear un algoritmo distinto al tradicional.
- La factorización QR que trabaja sobre bloques cuadrados se llama: factorización QR incremental, factorizacción QR *algorithm-by-blocks*, o *tiled QR factorization*.
- En este algoritmo, los bloques son siempre de la misma forma (cuadradados) y del mismo tamaño, independientemente de las dimensiones de la matriz y de la iteración del algoritmo.
- Además, dado que la matriz está almacenada en disco y las escrituras son más costosas, es conveniente emplear algoritmos *left-looking*: Se actualiza el bloque actual en función de todos los bloques anteriores.

Factorización QR Incremental o Factorización QR *algorithm-by-blocks* (Cont.)

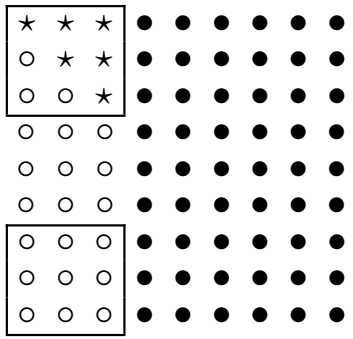
- El algoritmo que trabaja con bloques cuadrados es distinto al tradicional.



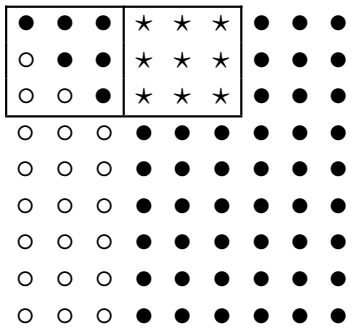
(1) After Compute_dense_QR(A_{00})



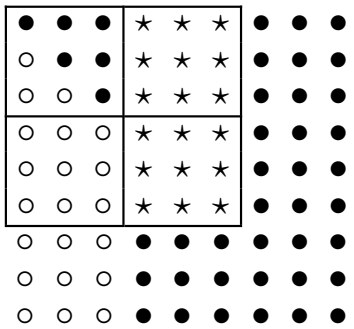
(2) After Compute_TD_QR(A_{00}, A_{10})



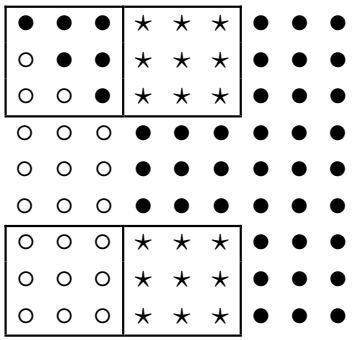
(3) After Compute_TD_QR(A_{00}, A_{20})



(4) After Apply_left_Qt_of_Dense_QR(A_{00}, A_{01})



(5) After Apply_left_Qt_of_TD_QR(A_{00}, A_{10} A_{01}, A_{11})

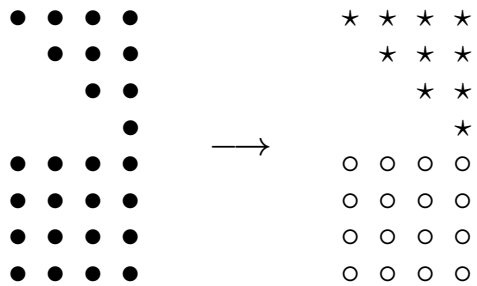


(6) After Apply_left_Qt_of_TD_QR(A_{00}, A_{20} A_{01}, A_{21})

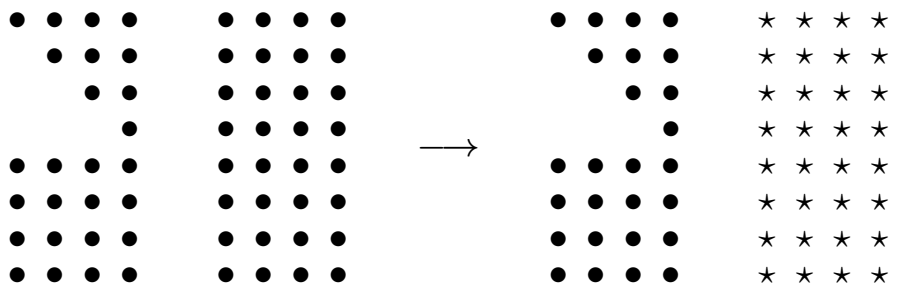
. . .

Núcleos Computacionales

- La factorización QR incremental requiere dos operaciones nuevas implementadas muy eficientemente (por bloques):
  1. `Compute_TD_QR`: Calcula la factorización QR de una matriz compuesta por una parte superior triangular (T) y una parte inferior densa (D).



  2. `Apply_left_Qt_of_TD_QR`: Aplica la TD QR anterior.



Tamaño de Bloque en la Factorización QR Incremental

- Todos los bloques (menos los extremos si el tamaño no es múltiplo) son de la misma dimensión: $n_b \times n_b$.
  - Coste de transferencia de cada bloque: $O(n_b^2)$ datos.
  - Coste computacional de cada bloque: $O(n_b^3)$ datos.
- Para tamaños de bloque grandes, el coste computacional aumenta muchísimo.
  - Si se duplica el tamaño del bloque, la cantidad de datos se multiplica por 4, pero el coste computacional se multiplica por 8.
  - Muy importante para minimizar el impacto de las transferencias.

Runtime

- Nuestra aplicación para la resolución de sistemas está basada en la biblioteca `libflame`.
- El *runtime* desarrollado es muy general y es capaz de ejecutar cualquier lista de tareas que trabaje con bloques cuadrados: factorización de Cholesky, LU incremental, QR incremental, UTV, producto de matrices, etc.

Runtime (Cont.)

- Funcionamiento del *runtime*:
  1. Al comenzar la ejecución, se crea una lista de tareas (que procesan bloques).

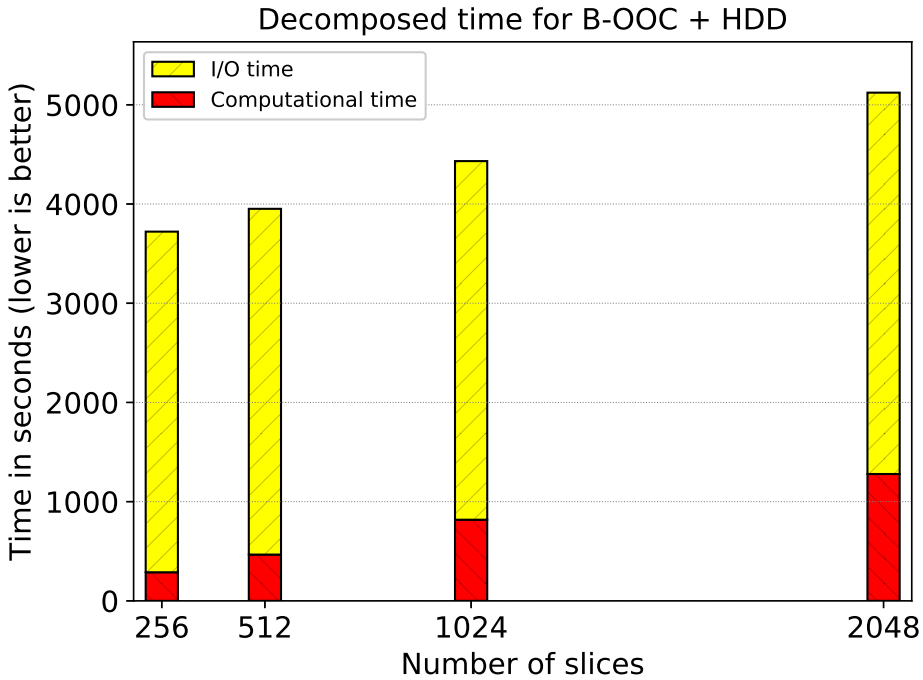
     Ejemplo de lista de tareas de la factorización QR incremental:

Operation	Out Operands	In Operands
Comp_dense_QR	A_{00} S_{00}	A_{00}
Comp_TD_QR	A_{00} A_{10} S_{10}	A_{00} A_{10}
Comp_TD_QR	A_{00} A_{20} S_{20}	A_{00} A_{20}
Apply_left_Qt_of_dense_QR	A_{01}	A_{00} S_{00} A_{01}
Apply_left_Qt_of_TD_QR	A_{01} A_{11}	A_{10} S_{10} A_{01} A_{11}
	$\vdots$	

  2. Después, se cede el control al *runtime*, el cual se encarga de ejecutar las tareas mediante:
     - Una antememoria en RAM de los bloques del disco.
     - El solapamiento de computación y E/S.

Primera Evaluación en un PC de Sobremesa

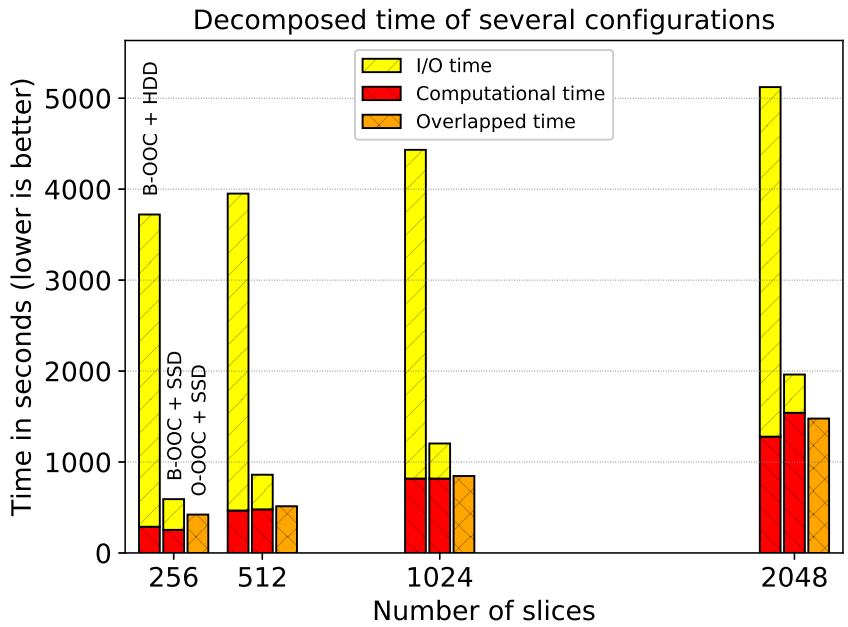
- Evaluación en un PC de sobremesa y un disco duro tradicional (giratorio):



- El número de slices es el número de imágenes que se calculan simultáneamente. Es igual al número de columnas de X y B en el sistema $AX = B$.
- ¿Cuál es el cuello de botella?

Mejora del PC con un SSD M.2

- Además del disco duro tradicional (barras de la izq.), como el cuello de botella es la E/S, se evalúa un disco duro SSD (barras centrales) y el solapamiento de cálculo y E/S (barras de la der.):

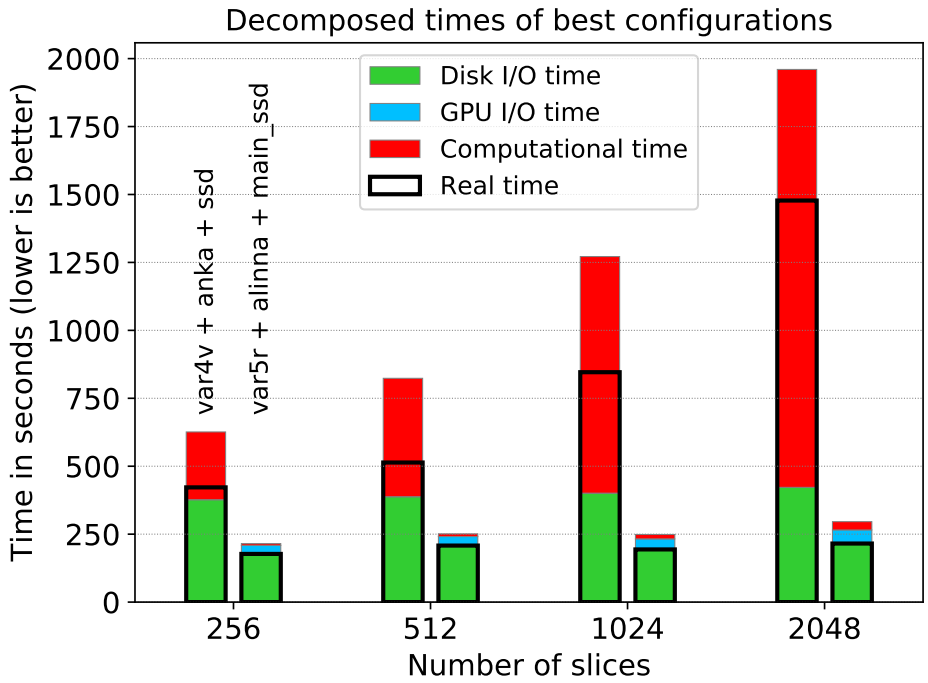


- ¿Cuál es el cuello de botella cuando se emplea un SSD?

Mejora con un Servidor con GPU y SSD M.2

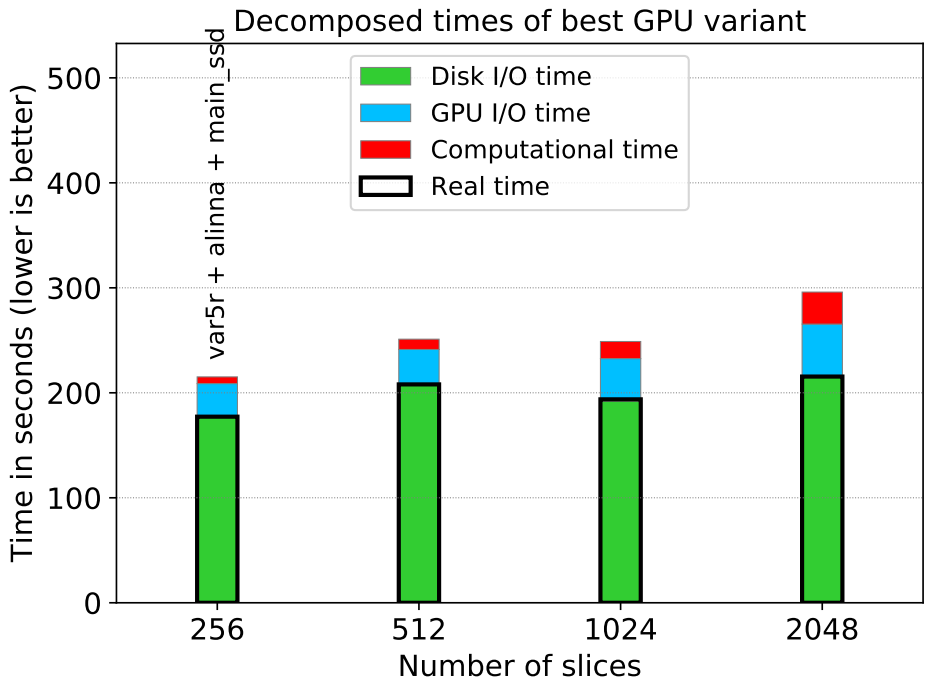
- Como el cuello de botella es la CPU, se evalúa un nuevo servidor con una GPU A100 (barras de la der.).

  Las barras de la izquierda corresponden a la mejor configuración de la transparencia anterior (CPU y disco SSD).



Mejora con un Servidor con GPU y SSD M.2 (Cont.)

- Esta gráfica corresponde a la mejor configuración anterior (GPU y disco SSD).



- ¿Cuántas imágenes por segundo se generan?
- ¿Cuál es el cuello de botella ahora con una GPU y un SSD M.2?

Mejora con un Servidor con GPU y SSD M.2 (Cont.)

- Dado que el cuello de botella ahora con una GPU y un SSD M.2 es la E/S, una altenativa sería la memoria Optane de Intel.

  Pero el producto ha sido cancelado.

Sistemas de Tamaño Enorme de Rango Deficiente

Resolución de Sistemas ENORMES de Rango Deficiente

- Si los datos NO caben en RAM, habrá que guardarlos en disco. Y los discos son bastante lentos.
- Si la matriz de coeficientes **tiene rango deficiente**, la resolución del problema lineal de cuadrados mínimos $\text{mín}_x ||Ax - b||_2$ puede basarse en:
  1. Factorización QR con pivot. de columnas: $AP = QR$
  2. Factorización SVD: $A = U\Sigma V^T$
- Ambas realizan muchas operaciones operaciones matriz-vector, con baja intensidad aritmética.
- Este tipo de operaciones requieren mucho movimiento de datos con respecto a la computación.
- Por tanto, obtienen muy bajo rendimiento.

Resolución de Sistemas ENORMES de Rango Deficiente (Cont.)

- Por suerte, puede venir en nuestra ayuda una nueva factorización lineal basada en el Algebra Lineal Randomizada:
  - Factorización randUTV: $A = UTV^*$
  - Coste: Unas 5,5 veces más costosa que la QR y también más costosa que la SVD, pero muy basada en BLAS-3.

Factorización randUTV de Matrices de Rango Deficiente

- La factorización randUTV de una matriz A de rango completo descompone la matriz A en el producto de matrices:

$$A = UTV^*,$$

donde T es triangular superior.

- La factorización randUTV de matrices de rango k $(k < n)$ genera un resultado de la siguiente forma:

$$A = UTV^T = U \begin{pmatrix} T_{11} & T_{12} \\ 0 & 0 \end{pmatrix} V^*,$$

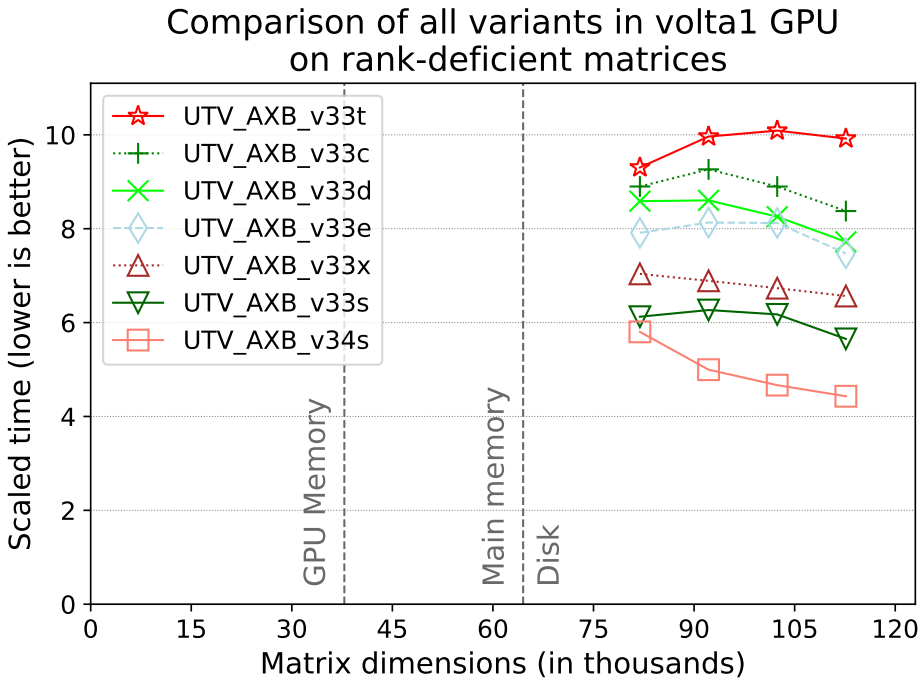
donde T_{11} es triangular superior y de dimensión $k \times k$.

Factorización randUTV de Matrices de Rango Deficiente (Cont.)

- Para poder resolver el sistema de ecuaciones, se debe anular el bloque T_{12} tras la factorización anterior aplicando transformaciones ortogonales adicionales.

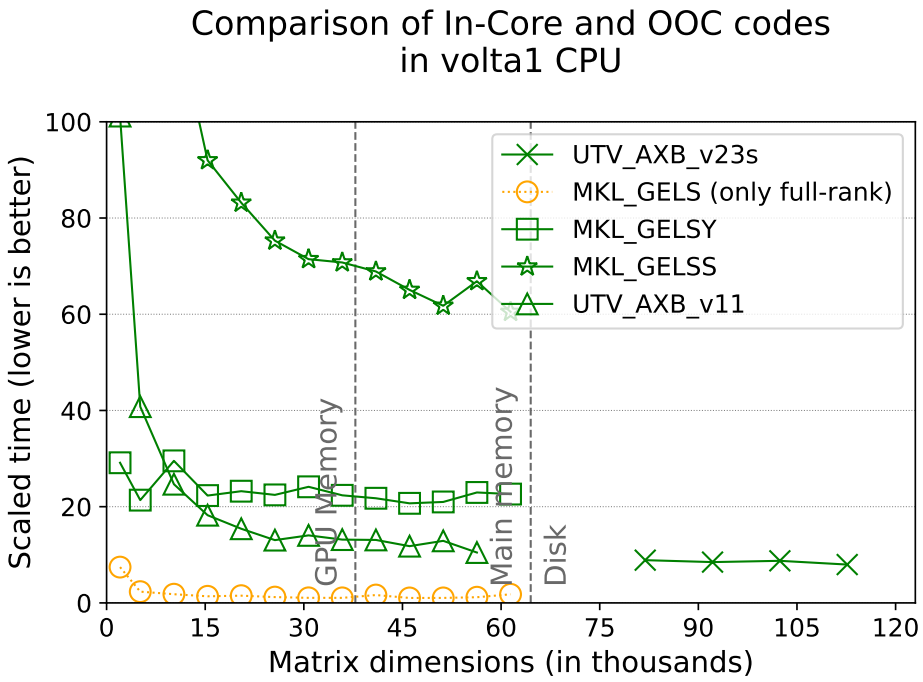
Prestaciones: Comparación de Variantes

- Comparación de algunas de nuestras variantes para GPU:



Prestaciones en CPU: Comparación *Incore* vs. OOC

- Comparación en CPU de códigos comerciales que trabajan con todos los datos en memoria central y nuestros códigos OOC (que trabajan con datos en disco):



Prestaciones en GPU: Comparación *Incore* vs. OOC

- Comparación en GPU de códigos comerciales que trabajan con todos los datos en memoria central y nuestros códigos OOC (que trabajan con datos en disco):