PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 11-03

Inclusión de estrategias de paralelismo al MOHID

Ignacio Barreto Pablo Ezzatti Mónica Fossati

2011

Inclusión de estrategias de paralelismo al MOHID
Barreto, Ignacio; Ezzatti, Pablo; Fossati, Mónica
ISSN 0797-6410
Reporte Técnico RT 11-03
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, abril de 2011

Inclusión de estrategias de paralelismo al MOHID

Ignacio Barreto[1], Pablo Ezzatti[1], Mónica Fossati[2]

[1] Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay
{ibarreto, pezzatti}@fing.edu.uy

[2] Instituto de Mecánica de Fluidos e Ingeniería Ambiental, Facultad de Ingeniería, Universidad de la República, Uruguay
{mfossati}@fing.edu.uy

04/2011

Palabras claves: HPC, descomposición de dominios, MOHID.

Resumen

En este documento se resumen diferentes diseños de extensión del framework propuestos con el objetivo de mejorar el desempeño computacional del modelo numérico hidrodinámico MOHID. Las versiones desarrolladas buscan aplicar la técnica de descomposición de dominios (DDM). La importancia de desarrollar una versión que incluya técnicas de DDM del modelo hidrodinámico radica en que actualmente el MOHID está siendo utilizado por investigadores perteneciente al grupo Hidráulica Fluvial y Marítima del Instituto de Mecánica de Fluidos e Ingeniería Ambiental (IMFIA) para simular diferentes procesos hidrodinámicos y de transporte de sustancias en el Río de la Plata.

Se presenta la descripción de tres diseños realizados aplicando la técnica de DDM. En cada diseño descrito se enumeran los diferentes cambios introducidos en el MOHID, qué funciones modificar, el formato de archivos de entrada utilizados para efectuar la descomposición y una descripción en alto nivel del funcionamiento de la última versión implementada.

En el resto del reporte se mencionan las evaluaciones preliminares realizadas en MOHID para lograr comprender el funcionamiento del modelo, se comentan varias consideraciones generales como el formato de archivo definido para la configuración de los dominios, los diferentes cambios a realizar en los archivos de compilación, en el código y en los archivos de entrada del modelo. Por último, se comentan diferentes mejoras a realizar sobre las propuestas.

Índice

1 Introducción

El modelo de simulación hidrodinámico MOHID es utilizado por investigadores del Instituto de Mecánica de los Fluidos e Ingeniería Ambiental (IMFIA) para el estudio del comportamiento del Río de la Plata. En la actualidad es de interés trabajar con simulaciones de grandes áreas que incluyen el Océano Atlántico y utilizar el MOHID en forma acoplada con modelos atmosféricos. El objetivo es desarrollar una herramienta operacional utilizando un esquema de modelos encajados. Esto implica por un lado calcular dominios de grandes dimensiones y por otro el cálculo en tiempo real de las ecuaciones gobernantes.

No es necesario profundizar en la importancia de contar con una herramienta operacional del Río de la Plata, sin embargo el esquema buscado trae aparejados grandes costos computacionales.

En base a lo mencionado anteriormente, el objetivo del trabajo es la inclusión de estrategias de computación de alto desempeño al MOHID. Buscando abatir los grandes costos computacionales mencionados.

En trabajos previos se estudió pormenorizadamente el MOHID [1] y se migró el modelo para ser utilizado en entornos con sistemas operativo LINUX [4].

En el reporte se presenta la descripción de tres diseños diferentes realizados aplicando la técnica de descomposición de dominios (DDM) [7]. Primero se mencionan las evaluaciones previas realizadas en la versión serial y la versión serial que utiliza submodelos encajados (teniendo en cuenta el valor de la variable *DT* tanto del modelo padre como de los submodelos hijos). Luego se presenta la secuencia de llamadas a los principales procedimientos y a qué módulos pertenecen. Esta misma secuencia, con algunas variantes, se ejecuta en las diferentes versiones de MOHID, incluso en las versiones desarrolladas. En cada diseño descrito se enumeran los cambios a ser introducidos en el modelo, qué funciones modificar, el formato de archivos de entrada utilizado para efectuar la descomposición y una descripción en alto nivel del funcionamiento de la última versión implementada. Por último, se enumeran las diferentes líneas a abordar en el futuro y las posibles mejoras a realizar en las propuestas.

2 Aplicación de paralelismo a MOHID

En esta sección se describen diferentes diseños preliminares realizados sobre el MOHID con el objetivo de incorporar estrategias de programación paralela bajo el paradigma de descomposición de dominios (DDM).

Uno de los objetivos del uso del MOHID es el modelado y simulación de la hidrodinámica de una gran área que abarca el Río de la Plata y una sección del Océano Atlántico, con el acople de modelos atmosféricos que hacen su aporte a las condiciones en la superficie del agua.

Los objetivos planteados en los distintos diseños son por un lado mejorar el desempeño computacional del modelo y por otro, escalar en la dimensión de los problemas a tratar, pudiendo abordar la resolución de escenarios que actualmente las limitantes de memoria RAM no lo permiten. Notar que con la tecnología actual, las simulaciones están limitadas (debido a la memoria RAM) por el tamaño del dominio a modelar y/o por la precisión con la cual se desea simular. El objetivo buscado en el proyecto es poder realizar simulaciones con dominios de mayores dimensiones o con un nivel de refinamiento mayor, en tiempos de ejecución aceptables.

Una forma de lograr el objetivo planteado en el párrafo anterior es mediante la división del dominio original en varios subdominios. Por ejemplo, ejecutando MOHID en varios nodos de cómputo diferentes y en cada uno procesando un submodelo como si fuera un único modelo global. Esta estrategia implica que en la etapa de construcción del dominio en memoria hay una etapa de descubrimiento de los dominios vecinos con los cuales se comparten datos necesarios para llevar a cabo la simulación en forma correcta. Los datos a compartir pertenecen a las variables simuladas durante la ejecución, por ejemplo el nivel de agua, las velocidades y los flujos, entre otros. En particular, cada subdominio envía los valores de dichas variables que se encuentran sobre la frontera que comparte con los subdominios vecinos. La comunicación entre los subdominios se hace a través de la biblioteca de pasaje de mensajes MPI permitiendo ejecutar un subdominio en un procesador en forma independiente de los demás subdominios. Esta manera de realizar la descomposición de dominios evita replicar la totalidad de los datos en los dominios involucrados logrando la capacidad de escalado en los dominios a simular.

2.1 Evaluaciones previas

Antes de avanzar en el diseño y desarrollo de versiones paralelas, se realizó una etapa de identificación de los procedimientos que insumen la mayor parte del tiempo de ejecución en cada paso de simulación de un modelo. El resultado de esta etapa permite enfocar los esfuerzos en mejorar las secciones del código críticas, facilitando así la mejora del desempeño computacional global de la simulación.

Se estudió la ejecución de MOHID en la versión serial sin submodelos encajados, la versión serial con submodelos encajados y la versión con submodelos encajados que utiliza MPI [1]. En cada una se analizo la diferencia entre los tiempos de ejecución, en especial, en lo que respecta al pasaje de datos entre el modelo padre y el/los submodelo/s encajado/s, qué datos se intercambian y que estructuras o matrices se utilizan en el caso de modelos encajados.

Cabe acotar que al resolver modelos encajados, el modelo hijo resuelve el mismo sistema de ecuaciones que el padre, o sea, calcula las mismas variables pero con distintas condiciones de borde. Si los cálculos se hacen en puntos diferentes al del dominio padre (la zona de simulación del dominio padre siempre incluye a la de los dominios hijos), se realiza una interpolación entre los puntos del dominio padre y el/los dominio/s encajado/s en cada paso de tiempo. El pasaje de las condiciones de borde se realiza desde el padre hacia el hijo a través de la función SubModelComunication. El DT (intervalo de tiempo de resolucion del sistema) de cada submodelo debe ser un divisor del DT del modelo padre, quedando como resultado de esa división los pasos que se realizan en el submodelo hijo, por cada paso que realiza el modelo padre. En el primer paso del modelo hijo se obtienen las condiciones de borde del padre. Esto tiene como consecuencia que el tiempo de ejecución del primer sub-paso es mayor que el tiempo de ejecución de los demás. Los detalles del análisis del tiempo de ejecución de las distintas versiones se encuentran en el reporte de Barreto et al [1].

2.2 Consideraciones generales

Si bien los prototipos que se implementaron fueron diseñados para tener tantos dominios como se quiera, fueron probados únicamente con 2 dominios generados a partir de la división según la dirección *j* del dominio original a simular. Por ejemplo, si el dominio original en el cual se ejecuta con la versión serial de MOHID tiene un tamaño (i,j) de (100,100), la versión paralela (de descomposición de dominios) tendría dos dominios, cada uno de tamaño (100,50).

Todos los diseños que se presentan a continuación, poseen una estructura de datos que es utilizada por cada dominio y que contiene varios datos necesarios para la ejecución de una simulación. Esta estructura es una lista en la que cada nodo contiene el *id* propio de MPI asignado en tiempo de ejecución, el *id* de MPI del dominio vecino, y un string que indica en qué lugar se encuentra el vecino respecto al dominio actual. Adicionalmente se tiene un archivo de descripción de la configuración de dominios llamado *datosVecinos_Num*, donde *Num* es el *id* de MPI que se le asignará en tiempo de ejecución.

El formato de este archivo es el siguiente:

IdDominio
Cantidad de vecinos
domv1, domv2, …..., etc
ubicaciónv1, ubicaciónv2, …. , etc

IdDominio: es un número entero positivo, que es independiente del *id* de MPI.
Cantidad de vecinos: indica la cantidad total de vecinos que tiene el dominio en cuestión.
domvX: es una lista de identificadores de dominio, la cantidad de elementos está determinada por el valor de cantidad de vecinos.
ubicaciónvX: es una cadena de caracteres indicando en dónde se ubica el dominio vecino. La cantidad de elementos depende del valor indicado en el campo cantidad de vecinos.

Se definieron 4 valores preestablecidos que permiten identificar la ubicación de los vecinos relativos al dominio que se está procesando. En esta cadena de caracteres cada

palabra está separada por un espacio en blanco y toma uno de los 4 valores preestablecidos, estos son: *Jprim*, *Jult*, *Iprim* e *Iult*, donde *Jprim* e *Iprim* indican que el dominio vecino se encuentra hacia los valores iniciales de *J* e *I* respectivamente; análogamente *Jult* e *Iult* indican que el vecino en el sentido ascendente de *J* e *I* respectivamente. Estas cadenas son de suma importancia porque indican en qué sentido y con qué valores de las matrices que estén involucradas hacer los envíos (MPI_Send) y obviamente las recepciones (MPI_Recv).

Otra consideración a realizar es en lo que respecta a la compilación. Se creó un nuevo archivo nix [5], siguiendo las ideas de tener una jerarquía de Makefiles para las distintas plataformas. El nuevo archivo se llama *nix_ddm*, en el cual se cambia el nombre del ejecutable, se agregó una nueva flag llamada _USE_DDM. Para poder utilizar el archivo se agregó "nix_ddm" en la línea PLAT del archivo Makefile.

Para el funcionamiento general de los distintos diseños se hicieron algunos cambios en algunas llamadas relacionadas con la escritura de archivos en el formato HDF5 [6]. En general, en el modelo original, estas llamadas tienen escrito explícitamente el nombre del archivo al que hacen referencia y no a través de una variable que contenga ese nombre, con lo cual no se independiza el programa de un nombre de archivo en particular. El cambio realizado es concatenarle a ese nombre de archivo un "_" y el *id* de MPI del dominio que está ejecutando ese proceso. La función que fue modificada en su invocación es: ConstructHDF5 la cual es invocada en ModuleWaterProperties, ModuleTurbulence, ModuleHydrodynamic y en ModuleInterfaceSedimentWater.

En el Main se incluye el acceso al archivo *datosVecinos_XX* que tiene la configuración de los dominios vecinos para el dominio que se ejecuta. Este archivo tiene como sufijo "_id" este sufijo se concatena en tiempo de ejecución.

Respecto a los archivos de entrada para la simulación se agregaron algunas líneas. En el archivo indicado en nomfich en la etiqueta IN_BATIM se modificó el nombre de archivo de batimetría, siendo el nombre de archivo sin el sufijo "_idMpi". En la etiqueta DISPQUAL, IN_TURB e IN_DAD3D se agregó "_id" a la clave TIME_SERIE_LOCATION seguido del archivo correspondiente a los puntos del dominio en donde MOHID, cada cierto tiempo, registra cómo evolucionan ciertas variables. También en el archivo indicado por DISPQUAL se concatenó "_id" en la clave DISCHARGES, al igual que en el archivo indicado por IN_DAD3D en la clave WATER_DISCHARGES. Esto se debe a que los valores del archivo en estos ejemplos de simulación (Ptos_generales.dat) deben estar coherentes con las coordenadas de los dominios involucrados en la descomposición de dominios. Respecto a la clave IN_BATIM, se modificó en el código, en ModuleModel dentro del procedimiento ConstructModel, la invocación a la función ReadFileName; el nombre de archivo tiene concatenado al final, el *id* de MPI del dominio que se está ejecutando. Se hizo lo mismo en la función ConstructHorizontalGrid perteneciente a ModuleHorizontalGrid. Siguiendo la idea anterior se modificó la etiqueta TIME_SERIE_LOCATION por TIME_SERIE_LOCATION_X, siendo *X* el *id* de MPI, esto se aplicó en la función Construct_Time_Serie en ModuleTurbulence, ModuleWaterProperties y en ModuleHydrodynamic.

2.3 Secuencia de ejecución

Para poder dar una idea de la ejecución del código en los diferentes diseños presentados se explica previamente la secuencia principal de invocación de la versión original desde el main, abstrayéndose de la implementación en si (Pseudocódigo 1). Se muestran las secuencias de llamadas a diferentes procedimientos y a qué módulo pertenece. Luego dentro de cada diseño propuesto se explica qué cambios o mejoras fueron realizadas.

```
ConstructMohidWater
     ConstructModelList
     ConstructModel (para todos los dominios, ModuleModel)

     ConstructFatherGrid (en caso que el modelo sea un
                     subdominio construye la grilla del padre)
     GetHydroNeedsFather

     GetWaterNeedsFather (se verifica cuál dominio requiere del
                     padre condiciones tanto hidrodinámicas como
                     de propiedades del agua)

     SetHydroFather (ModuleHydrodynamic) (si existen subdominios
                                          encajados)

     SetWaterPropFather (ModuleWaterProperties) (si existen
                                 subdominios encajados)
ModifyMohidWater
     UpdateTimeAndMapping (ModuleModel)
     SubModelComunication
     RunModel (ModuleModel)
          Turbulence (ModuleTurbulence)
          Modify_Hydrodynamic (ModuleHydrodynamic)
          WaterProperties_Evolution (ModuleWaterProperties)
KillMohidWater
```

Pseudocódigo 1 - Llamadas principales desde Main.F90.

En el Pseudocódigo 2 se muestra en detalle las llamadas que se hacen dentro de la función Modify_Hydrodynamic y sus sucesivas llamadas relevantes dentro de cada una de las funciones que se invocan.

```
Modify_Hydrodynamic (ModuleHydrodynamic)
   One_Iteration
        MomentumMassConservation
           Leendertse_Scheme
          MaintainDirection
          Explicit_Forces
          Modify_HorizontalWaterFlow
          ChangeDirection
          Bottom_Boundary
          Explicit_Forces
          Compute_WaterLevel
               WaterLevel_BarotropicPressure
               WaterLevel_ExplicitForces
               WaterLevel_WaterFluxes
               THOMAS_2D (ModuleFunctions)
               WaterLevelRelaxation
               WaterLevelRelaxationAltimetry
               WaterLevelCorrection
          Compute_Velocity
               SetValueMatrix
               Velocity_ExplicitForces
               VelVerticalDiffusionBondaries
               Velocity_VerticalDiffusion
               Velocity_VerticalAdvection
               ComputeAdvectionFace
               Velocity_WaveStress
               THOMAS_3D(NoOpenMP)(ModuleFunctions)
                    THOMAS_3D_i1_j0
                    THOMAS_3D_i0_j1
               InstantMixingSmallDepths
               Velocity_OpenBoundary
     New_Geometry
```

Pseudocódigo 2 - Llamadas desde función Modify_Hydrodynamic.

Otra consideración general a todos los diseños es que en la etapa inicial de construcción del modelo, cada subdominio realiza una serie de envíos y recepciones (Sends y Receives) para obtener datos del *id* de MPI de los vecinos para así reconocer cuál es la configuración de la descomposición del dominio original. El pseudocódigo 3 presenta la etapa de construcción de la información de los vecinos, la cual es realizada por la función ConstruirVecinos. Esta función busca, para cada subdominio, reconocer quiénes son sus subdominios vecinos.

```
ConstructMohidWater

ConstruirVecinos
     Lectura de datosVecinos_XX

     Para todos los procesos de MPI hacer
          Send (id_MPI, id_Dominio)
     Fin hacer

     i=1
     Mientras i<=Cant vecinos del proceso actual
          Receive (id_MPI, id_Dominio)

          buscarIDVecino(id_Dominio, posiciónVector)

          Si posiciónVector <> -1
               vecinos (posiciónVector)%mpiID = id_MPI
               i = i +1
          Fin si
     Fin Mientras
ModifyMohidWater
KillMohidWater
```

Pseudocódigo 3 – Construcción de vecindad en Main.F90.

2.4 Construcción de los dominios

La función ConstruirVecinos tiene como cometido inicializar la estructura *T_Vecinos* y hacer el reconocimiento de los dominios vecinos almacenando los datos que correspondan para poder identificarlos. Esto se realiza para cada dominio.

La manera en que se lleva a cabo esta tarea se puede observar en el pseudocódigo 3 en las llamadas que se hacen dentro de ConstruirVecinos. Allí se lee el archivo *datosVecinos_XX*, donde *XX* es el *id* de MPI que corresponde al proceso que se está ejecutando. Luego se hace un broadcast hacia los demás procesos enviando como datos el *id* de MPI asignado y el *id* de dominio leído del archivo *datos_Vecinos_XX*. Luego se pasa a una etapa de recepción de datos en la cual el proceso actual recibe los datos de todos los procesos. Se invoca a la función buscarVecino para buscar si el *id* de dominio recibido coincide con algún *id* de dominio leído de *datos_VecinosXX*. En caso afirmativo se asigna en el vector de vecinos el *id* de MPI recibido en el lugar que corresponda. Estos pasos se hacen para todos los procesos de MPI que se ejecutan.

3 Prototipos realizados

A continuación se describen los diferentes prototipos realizados con el objetivo de mejorar el desempeño computacional en las simulaciones que realiza el Grupo de hidráulica fluvial y marítima del IMFIA utilizando el modelo MOHID.

3.1 Dominio

El dominio original con el que se realizaron las pruebas para comprobar las mejoras computacionales es una grilla de 102 x 101, como se muestra en la figura 1 (filas, columnas), la cual no contiene ningún volumen que se corresponda a tierra, es decir, todos los elementos se corresponden a agua. Los dominios resultantes de la descomposición son uno de 102 x 51(Dom0) y el otro de 102 x 50 (Dom1) como se muestra en la figura 2.

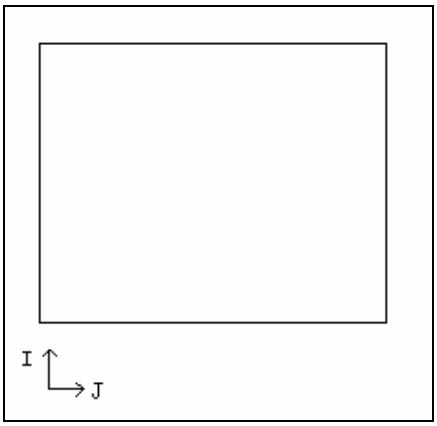


Figura 1 - Dominio original.

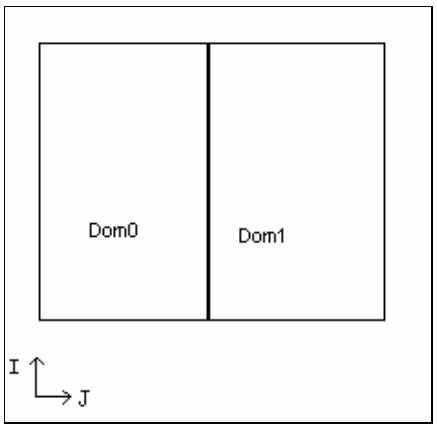


Figura 2 – Dominio original particionado.

3.2 Primera versión

La idea fundamental de este primer diseño paralelo, es que al finalizar cada paso de simulación se envían y reciben una subsección de todas las matrices que se encuentran dentro del tipo de datos principal definido en ModuleHydrodynamic y ModuleWaterProperties. Se incluyen también las matrices anidadas dentro de toda esa estructura. Estos tipos principales son T_Hydrodynamic y T_WaterProperties. Los datos que se intercambian entre los subdominios vecinos no son las matrices completas, sino que se envía y recibe la columna o fila del propio dominio que esté más cerca del dominio vecino.

Para realizar la transferencia de datos se crearon varias funciones:

ModuleHydrodynamic

CalcularTamanio
CalcularTamanioP2
SendDatosDDM
SendDatosDDMP2

ArmarPack
ArmarPackP2
ReceiveDatosDDM
ReceiveDatosDDMP2
DesarmarPack
DesarmarPackP2

ModuleWaterProperties

CalcularTamanioWaterProp
SendDatosDDMWaterProp
ArmarPackWaterProp
ReceiveDatosDDMWaterProp
DesarmarPackWaterProp

Dentro de ModuleHydrodynamic, SendDatosDDM y SendDatosDDMP2 son las funciones encargadas del envío de las matrices, agrupándolas previamente en un MPI_Pack. Se dividió en dos funciones y cada una de estas dos funciones invoca a ArmarPack y ArmarPackP2, las cuales van armando el MPI_Pack para ser enviado luego.

En forma análoga están las funciones ReceiveDatosDDM y ReceiveDatosDDMP2 conteniendo la función DesarmarPack y DesarmarPackP2 respectivamente, en donde se asigna lo recibido en la matriz correspondiente de la estructura T_Hydrodynamic.
Respecto al módulo ModuleWaterProperties, para hacer el envío y recepción de datos se siguió la misma idea realizada en ModuleHydrodynamic.

El pseudocódigo general de este diseño se presenta en el pseudocódigo 4. Las modificaciones fueron realizadas en el archivo Main.F90 dentro de la función ModifyMohidWater:

```
Dentro del loop principal de la función se agregó:
CalcularTamanio
CalcularTamanioP2
CalcularTamanioWaterProp
Para cada vecino del dominio que se está ejecutando hacer:
          SendDatosDDM
          SendDatosDDMP2
          SendDatosDDMWaterProp
Fin hacer
Para cada vecino del dominio que se está ejecutando hacer:
          ReceiveDatosDDM
          ReceiveDatosDDMP2
          ReceiveDatosDDMWaterProp
Fin hacer

Luego sigue ejecutándose ModifyMohidWater (p/cada dominio).
```

Pseudocódigo 4 – primer diseño de DDM.

Al comparar los resultados numéricos generados por esta versión y la versión serial se vio que se generaban ciertas diferencias. Esto se debe a que no es correcto enviar y recibir todos los datos en la frontera de la estructura T_Hydrodynamic porque en realidad se estarían copiando valores que en el modelo serial no son utilizados para los cálculos de las variables simuladas. Esto significa que no es correcto hacer esa asignación directa e incluso tampoco lo es hacer un promedio del valor actual con el valor recibido por parte del dominio vecino. Por otro lado hay muchas matrices las cuales no afectan los cálculos de las variables simuladas en el dominio vecino, dicho de otra manera, esas matrices son propias del dominio que representa e independiente de los dominios vecinos.

3.3 Submodelos encajados

La segunda versión investigada, se basó en intentar emular la forma en la cual se realiza, y qué matrices están involucradas, en el pasaje de datos cuando se ejecuta MOHID en su versión serial con submodelos encajados. En dicha versión serial en cada paso del loop principal del Main se actualizan, o calculan, primero todos los subdominios. Esto quiere decir que cada hijo obtiene los valores del padre o hace una interpolación de las variables a calcular. Esto se hace en la función SetHydroFather que se encuentra dentro de SubModelComunication.

A continuación se presentan las matrices que representan los valores del dominio padre a partir de las cuales cada subdominio hijo puede interpolar los valores de la frontera de su dominio;

OpenPoints3D (es una matriz que se obtiene invocando a la función GetOpenPoints3D pasándole el IdMap, esta matriz se crea en el ModuleHorizontalMap)

ComputeFacesU3D
ComputeFacesV3D (se obtienen invocando GetComputeFaces3D con el IdMap)

DUZ
DVZ (se obtienen invocando GetGeometryDistances con el IdGeometry)

Me%Velocity%Horizontal%U%New
Me%Velocity%Horizontal%V%New

Me%WaterFluxes%X
Me%WaterFluxes%Y

Me%WaterLevel%New

Para seguir profundizando en la secuencia de llamadas en el pasaje de datos en la versión serial con submodelos encajados se menciona la secuencia de llamadas de las funciones SetHydroFather y SetWaterPropFather, de los módulos ModuleHydrodynamic y ModuleWaterProperties respectivamente.

```
SetHydroFather
     ReadNextOrInitialField
          ReadLockFather
          InterpolRegularGrid (con diferentes parámetros en
                               la invocación)
               InterpolPoint
          ReadUnLockFather
     ActualizeSubModelValues
          ReadLockSon

SetWaterPropFather
ReadNextOrInitialField
     ActualizeSubModelValues
```

Pseudocódigo 5 – Secuencia de llamadas para el pasaje de datos padre-hijo con encajados.

La función InterpolRegularGrid (ModuleFunctions) es la función que realiza la interpolación de los valores del dominio padre respecto de la grilla hija o subdominio encajado. Luego de realizado, se actualizan los valores en el subdominio hijo en ciertas matrices mencionadas anteriormente en la función ActualizeSubModelValues. Ocurre de manera similar en la función ActualizeSubModelValues de ModuleWaterProperties. En el Anexo A se muestra un detalle de los cálculos que se realizan en la función ActualizeSubModelValues dentro de ModuleHydrodynamic**.** En el Anexo B se presenta la secuencia de llamadas de las dos funciones más importantes que ocurren dentro de la función SetWaterPropFather al igual que en el módulo ModuleHydrodynamic estas funciones son ReadNextOrInitialField y ActualizeSubModelValues. En los anexos A y B se muestran lo más relevante de cada función. En el caso del Anexo A es respecto a ModuleHydrodynamic, pudiéndose apreciar qué matrices son accedidas en cada función, lo cual es un indicador de las matrices relevantes para la simulación.

Esta opción de diseño fue descartada por el hecho de que es muy difícil simular una interpolación entre los dominios debido a dos razones. Primero, el dominio encajado está contenido totalmente en el dominio padre, por lo cual la interpolación de valores de las matrices involucradas, entre los puntos de la grilla padre e hijo, se hace en forma directa. Segundo, la idea de la descomposición de dominios es dividir al domino original en varias partes, pueden solaparse en alguna región o no, pero nunca un dominio puede estar contenido en otro, lo cual hace difícil poder emular el pasaje de datos entre estos dominios de la misma forma que se hace con dominios encajados.

Un aporte positivo del proceso de investigación sobre este diseño es que permitió la identificación más detallada de las matrices involucradas en el pasaje de datos entre el modelo padre y los submodelos hijos en la versión de submodelos encajados de MOHID.

3.4 Versión basado en el algoritmo de Thomas

Esta tercera versión se enfoca en identificar cuáles son las matrices más relevantes durante una simulación. Como punto de partida se evaluó con las matrices que se utilizan para el pasaje de datos de los dominios encajados. Estas matrices en ModuleHydrodynamic son:

Me%Velocity%Horizontal%U%New
Me%Velocity%Horizontal%V%New

Me%WaterFluxes%X
Me%WaterFluxes%Y

Me%WaterLevel%New

El primer estudio fue sobre la matriz *Me%WaterLevel%New*, que es la que contiene los valores del nivel del agua; esta matriz está relacionada en el sentido de que se calcula a partir de otras matrices, como:

DCoef_2D → *Me%Coef%D2%D*
ECoef_2D → *Me%Coef%D2%E*
FCoef_2D → *Me%Coef%D2%F*
TiCoef_2D → *Me%Coef%D2%Ti*

Donde A→B significa que A es un puntero a la variable B.
La función que calcula esta matriz se encuentra en ModuleFunctions, THOMAS_2D. Esta función es central en la resolución del tercer diseño propuesto ya que de la matriz Me%WaterLevel%New dependen muchas matrices, como por ejemplo las matrices de velocidades horizontales y las matrices que representan a los flujos. Se analizó este algoritmo para comprobar si los valores que calculaba dependen o no del tamaño de la matriz (Me%WaterLevel%New) que está relacionado con el tamaño del dominio. Luego de analizado el algoritmo y de acuerdo a cómo se hacen los cálculos se concluyo que la función THOMAS_2D no depende del tamaño del dominio.

3.4.1 Algoritmo de Thomas

En esta sección se muestra el código del algoritmo de Thomas y la variante desarrollada en MOHID a través de la función THOMAS_2D. Se marcan las secciones relevantes para luego explicar en detalle su funcionamiento y por último especificar los cambios introducidos en esta función.

El Algoritmo de Thomas [2] resuelve un sistema lineal tri-diagonal ($Ax = b$); es también conocido como algoritmo de matriz tri-diagonal (TDMA). Este método es una variante simplificada del método de factorización LU y consiste en reducir el sistema tri-diagonal a dos sistemas bi-diagonales superior e inferior respectivamente. En el pseudocódigo 6 se muestra las etapas de este algoritmo.

```
Etapas:
     Descomposición
     Sustitución hacia adelante
     Sustitución hacia atrás
```

Pseudocódigo 6 – Algoritmo de Thomas.

- Descomposición:

```
DO k=2, n
     e_k=e_k/f_k-1
     f_k=f_k-e_k*g_k-1
END DO
```

- Sustitución hacia Adelante:

```
DO k=2, n
     b_k=b_k-e_k*b_k-1
END DO
```

- Sustitución hacia Atrás:

```
X_n=b_n/f_n
DO k=n-1,1,-1
     x_k=(b_k-g_k*x_k+1)/f_k
END DO
```

La variante desarrollada en MOHID se encuentra en ModuleFunctions (perteneciente a la biblioteca Mohid_Base_1) en la función THOMAS_2D. En vez de tener una matriz tridiagonal de dimensión MxN o tres vectores de largo MxN la variante utiliza tres matrices de tamaño MxN. Estas tres matrices están representadas a través de las variables *DCoef_2D, ECoef_2D y FCoef_2D* el vector independiente *b* está representado por la matriz *TiCoef_2D*.

La explicación del código se hará mostrando los valores que se asignan cuando se ejecuta THOMAS_2D para el dominio "Dom0" (el de "la izquierda").

Las variables que indican el tamaño del dominio y que son de control para el algoritmo son:
IJmin, *IJmax*, *JImin*, *JImax*, *di* y *dj*. Consideramos el caso en el cual están asignados de esta forma:

IJmin	Ijmax	JImin	JImax	di	dj
1	102	1	51	0	1

Nota: lo que está entre paréntesis rectos, son acotaciones y comentarios para ejemplificar el algoritmo con valores reales

```
Do IJ = IJmin, IJmax
        [1]      [102]

Inicialización1   I = IJ * dj  +  di  [→ I = IJ ]
                  J = IJ * di  +  dj  [→ J = dj = 1]

                  VECW(JImin) = -FCoef_2D (I, J) / ECoef_2D (I, J)
                  VECG (JImin) = TiCoef_2D (I, J) / ECoef_2D (I, J)

                  Do JI = JImin+1,  JImax+1
                            [2]              [52]
                          I = IJ * dj  +  JI * di      [→ I = IJ]
Loop A                    J = IJ * di  +  JI * dj      [→ J = JI]
                  VECW(JImin) = -FCoef_2D (I,J) / (ECoef_2D(I,J)+DCoef_2D(I,J)
*VECW(JI-1))

                  VECG (JImin) = (TiCoef_2D (I, J) – DCoef_2D(I, J)) * VECG(JI-1)) /
                                 (ECoef_2D (I, J)  + DCoef_2D(I, J) * VECW(JI-1))
                  enddo

Inicialización2   I = IJ * dj + (JImax+1) * di      [→ I = IJ]
                  J = IJ * di + (JImax+1) * dj      [→ J = JImax+1 = 52]
                  ANSWER (I, J) = VECG(JImax+1)

                  Do II = JImin+1, JImax+1
                          [2]              [52]
                          MM = JImax + 2 – II       [→MM= 53-II]
Loop B                    I = IJ *dj + MM * di      [→ I = IJ]
                          J = IJ *di + MM * dj      [→ J = MM]

                          ANSWER(I, J) = VECW(MM) * ANSWER(I+di, J+dj) +
VECG(MM)
                  enddo
enddo
```

Es importante remarcar que entre un paso de simulación y el siguiente se intercambian los valores *di* con *dj* y también *IJmax* con *Jimax;* tal cual se muestra en la tabla 1. En realidad lo que se invierte es el sentido en el que se recorren las matrices *DCoef_2D*, *ECoef_2D*, *FCoef_2D*, *TiCoef_2D* (las cuales de aquí en adelante se simplifcan em *D*, *E*, *F* y *Ti)*. Las variables *di* y *dj* se invierten debido a la invocación de la función ChangeDirection.

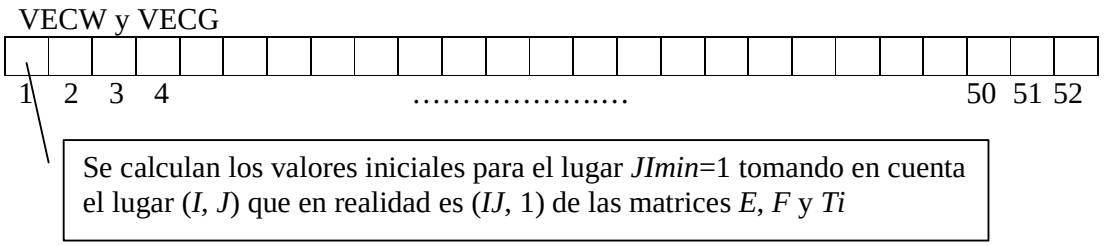
Pasos de simulación	IJmin	IJmax	JImin	JImax	di	dj
Paso t	1	102	1	51	0	1
Paso t + 1	1	51	1	102	1	0

Tabla 1 - Valores asignados para los tamaños de las matrices y vectores.

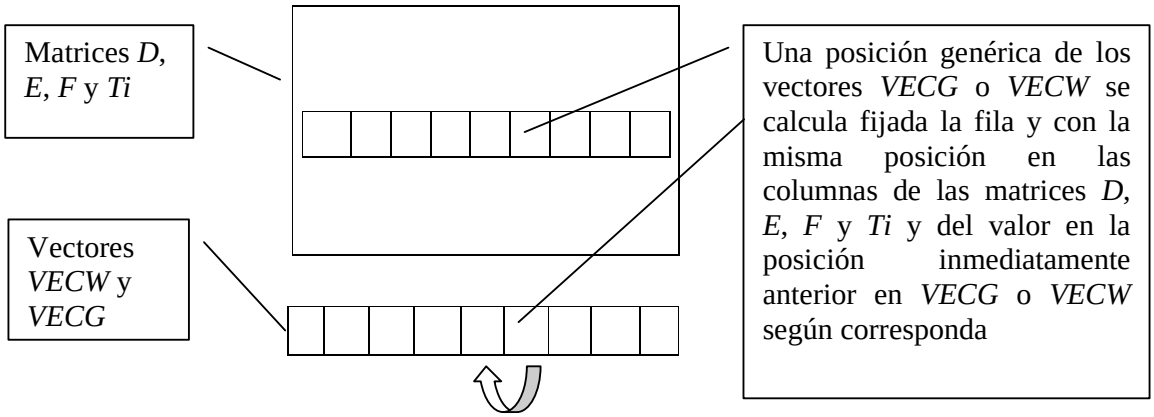
3.4.1.1 Implementación del Algoritmo de Thomas en MOHID

A continuación se mostrará gráficamente cómo funciona la implementación en MOHID del algoritmo de Thomas.

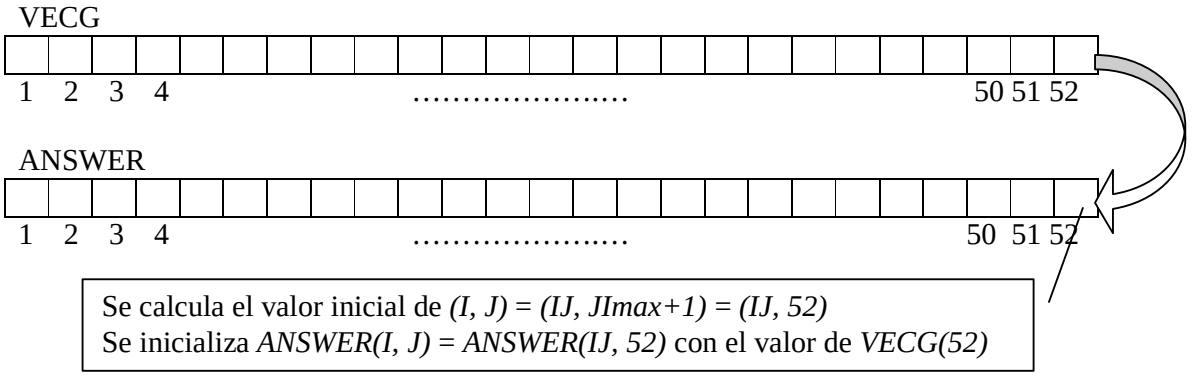
En "**inicialización1**" asigna los valores de *I* y *J*, a partir de estos valores se inicializan los vectores *VECG* y *VECW*.



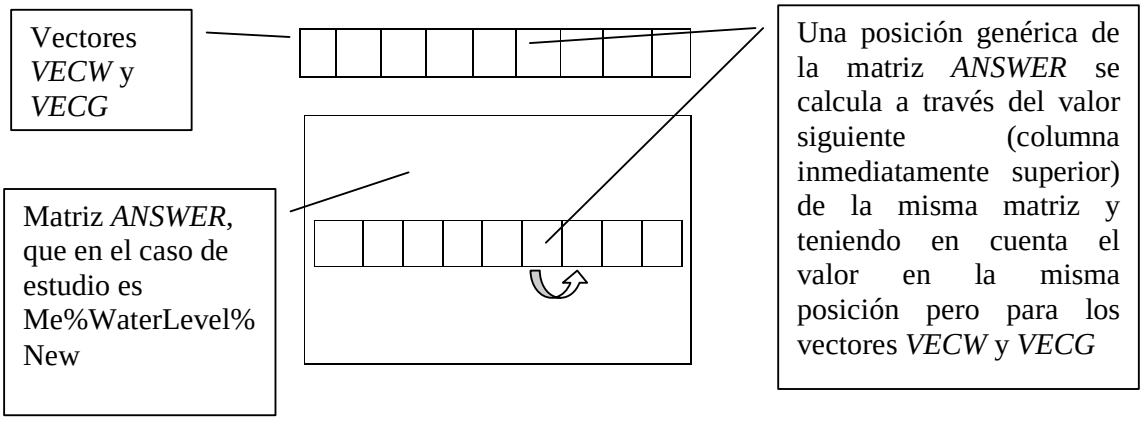
La etapa siguiente es la que se denominó "**Loop A**", en la cual se va calculando las posiciones restantes en los vectores *VECW* y *VECG*, dejando fija la fila y recorriendo las columnas de las matrices *D*, *E*, *F* y *Ti*.



Al terminar el Loop A ya están calculados todos los valores de *VECG* y *VECW*. El siguiente paso es "**inicialización2**"



El siguiente paso es "**Loop B**" que es similar al Loop A. Algunas variantes son: que las posiciones van decreciendo, la variable de control del Loop B es *II* y en el ejemplo presentado toma los valores entre 2 y 52 (en forma ascendente). En el caso de la variable *MM* toma los valores entre 51 y 1 (en forma decreciente). En el ejemplo la fila de la matriz *ANSWER* está fija y lo que va variando es la columna, desde el valor superior de columna hacia la menor.



Luego de terminado el Loop B está calculada toda una fila de la matriz *ANSWER*, a través del loop general, que lo lleva a cabo la variable de control *IJ*.

3.4.1.2 Modificación realizada en el algoritmo de Thomas de MOHID

Según el detalle presentado de cómo funciona esta variante del algoritmo de Thomas se observa que el cálculo de la matriz resultado es independiente del tamaño del dominio y que el cálculo de una celda solamente está relacionada con el valor de la celda anterior o de la celda siguiente, dependiendo en dónde se encuentre el algoritmo, si en el Loop A o en el Loop B. La modificación propuesta apunta a dividir el dominio original en varios subdominios, en principio dos subdominios, y que comparten la simulación en forma paralela. Los pasos que se presentan a continuación son las modificaciones realizadas en las distintas funciones de los diferentes Módulos de MOHID para luego poder hacer efectivas las modificaciones en la función THOMAS_2D:

1- La definición del tipo Vecinos se realiza en ModuleGlobalData. En las versiones previas esto se encontraba en Main.

2- En Main dentro de ConstructMohidWater se cambió la invocación de la función ConstructModel. Se le agregaron dos parámetros, la configuración de los vecinos del dominio actual y la cantidad de vecinos.

3- En ModuleModel dentro de ConstructModel se invoca a StartHydrodynamic pasándole como parámetros la configuración de vecinos y la cantidad de vecinos.

4- Se modificó la función StartHydrodynamic (ModuleHydrodynamic), si está presente la bandera _USE_DDM, se asigna una variable global que es la "ubicación". Además, al momento de realizar la invocación de la función

THOMAS_2D dependiendo de la presencia o no de la bandera se invoca con parámetros diferentes _USE_DDM.

5- Dentro de ModuleHydrodynamic en la función Compute_WaterLevel, que es donde se encuentra la invocación a THOMAS_2D, antes de esta invocación lo que se hace es obtener entre los vecinos del dominio actual, 4 vectores que corresponden a una columna o una fila, la segunda o la penúltima dependiendo de la ubicación del vecino, de las matrices *D*, *E*, *F*, *Ti*. Estos vectores son usados en la invocación de la función THOMAS_2D. Las funciones que realizan este envío y recepción de datos son SendDatosDDMThomas2D y ReceiveDatosDDMThomas2D respectivamente. La primera fila o columna y la última fila o columna de estas matrices (*D*, *E*, *F* y *Ti*), tienen almacenadas valores por defecto que indican en MOHID que son bordes del dominio.

6- Se cambia el encabezado de la función THOMAS_2D agregándose un parámetro que indica los vecinos del dominio actual para saber en dónde se tiene que fijar para recibir y enviar datos. Además de los vectores auxiliares, como se explicó en el punto anterior *D, E, F* y *Ti*.

La siguiente secuencia explica el funcionamiento de la función THOMAS_2D modificada. En la siguiente sección se muestra el código original marcando los cambios introducidos.

1- El dominio que se encuentra “a la derecha” está esperando a recibir datos desde el dominio “de la izquierda”.

2- Se empieza entonces por el dominio “de la izquierda” con la etapa inicialización1 y con el Loop A. La simulación del dominio “de la izquierda”, al terminar el Loop A calcula un valor ficticio que no se almacena en el dominio actual sino que se envía al dominio vecino. El valor ficticio es equivalente a calcular como si el dominio actual tuviera un lugar extra en *VECW* y *VECG*, y por consiguiente una fila o columna extra, dependiendo el caso, en la matriz resultado *ANSWER*. Para calcular estos valores se utilizan los vectores auxiliares ingresados por parámetro y los valores que se encuentran en la posición *JImax*-1 de los vectores *VECW* y *VECG*.

3- El dominio “de la izquierda” después de enviar los valores, calculados como se explicó en el punto anterior, se queda esperando recibir datos del dominio “de la derecha”.

4- El dominio “de la derecha”, que estaba esperando, recibe los datos del dominio “de la izquierda”. Estos dos valores hacen las veces de inicialización 1, pero no con los valores locales, sino con los valores que obtiene del dominio vecino.

5- El dominio de la derecha ejecuta Loop A. Termina este loop y prosigue con la inicialización 2 y con el Loop B.

6- Al terminar el Loop B el dominio “de la derecha” envía el valor de la matriz *ANSWER* (*IJ*, *JImin*) hacia el dominio “de la izquierda”, que estaba esperando recibir este dato. El dominio “de la derecha” sigue con su simulación.

7- El dominio “de la izquierda” recibe este valor y continúa el algoritmo normalmente.

3.4.1.3 Algoritmo de Thomas modificado

A continuación se muestra el código del algoritmo de Thomas incluyendo las modificaciones realizadas a ese código.

```
                Do IJ = IJmin, IJmax
                        [1]        [102]

inicialización1 {       I = IJ * dj  +  di  [→ I = IJ ]
                        J = IJ * di  +  dj  [→ J = dj = 1]

                        VECW(JImin) = -FCoef_2D (I, J) / ECoef_2D (I, J)
                        VECG (JImin) = TiCoef_2D (I, J) / ECoef_2D (I, J)

                        Si di = = 0 y el vecino se encuentra en “Jprim”  entonces

Modificación 1 .                MPI_Recv (valor1, mpiIDVecino)

Modificación 1 /                MPI_Recv (valor2, mpiIDVecino)

                                VECW(JImin) = -FAux (IJ,1)/(EAux(IJ,1) + DAux(IJ,1) * valor1)

                                VECG(JImin) = (TiAux(IJ,1) - DAux(IJ,1) * valor2) / &
                                              (EAux(IJ,1) + DAux(IJ,1) * valor1)
                        Fin

                        Do JI = JImin+1,  JImax+1
                                [2]               [52]

                                I = IJ * dj  +  JI * di        [→ I = IJ]
                                J = IJ * di  +  JI * dj        [→ J = JI]

Loop A                  VECW(JImin) = -FCoef_2D (I,J) / (ECoef_2D(I,J) + DCoef_2D(I,J)
                                      *VECW(JI-1))

                                VECG (JImin) = (TiCoef_2D (I, J) – DCoef_2D(I, J)) * VECG(JI-1))
                        /
                                              (ECoef_2D (I, J)  + DCoef_2D(I, J) * VECW(JI-1))
                        enddo
inicialización2 {       I = IJ * dj + (JImax+1) * di        [→ I = IJ]
                        J = IJ * di + (JImax+1) * dj        [→ J = JImax+1 = 52]
                        ANSWER (I, J) = VECG(JImax+1)
```

Modificación 2

```
Si di = = 0 y el vecino se encuentra en Jult   entonces
        valor1   = -FAux (IJ,1)/(EAux(IJ,1) + DAux(IJ,1) * VECW(JImax-1))
        valor2   = (TiAux(IJ,1) - DAux(IJ,1) * VECG(JImax-1)) / &
                   (EAux(IJ,1) + DAux(IJ,1) * VECW(JImax-1))

        MPI_Send (valor1, mpiIDVecino)
        MPI_Send (valor2, mpiIDVecino)
        MPI_Recv (valor3, mpiIDVecino)
        VECW(JImax) = valor1
        VECG(JImax) = valor2
        ANSWER(I,J) = valor3
Fin
```

Loop B

```
Do II = JImin+1, JImax+1
        [2]                [52]

        MM = JImax + 2 – II      [→MM= 53-II]
        I = IJ *dj + MM * di     [→ I = IJ]
        J = IJ *di + MM * dj     [→ J = MM]

        ANSWER(I, J) = VECW(MM) * ANSWER(I+di, J+dj) +
        VECG(MM)
enddo
```

Modificación 3

```
Si di= = 0 y el vecino se encuentra en Jprim

        valor3 = ANSWER(IJ,IJmin)

        MPI_Send (valor3, mpiIDVecino)

enddo
```

```
enddo
```

3.4.2 Consideraciones de la versión basada en Thomas

Esta versión de modificación del algoritmo de Thomas se probó solamente para dos dominios que son la división de un dominio original de tamaño 102 x 101, siendo los dominios generados de tamaño 102 x 51 y 102 x 50, o sea, esta división se hizo respecto a las columnas de la grilla.

Al inicio de la simulación los valores de la matriz WaterLevelNew son bastante similares a los valores que se encuentran en la posición correspondiente en el modelo serial. Sin embargo, a medida que la simulación avanza, a partir de cierto paso, estos valores de la matriz WaterLevelNew comienzan a variar (a ser diferentes respecto a los valores que se encuentran en las posiciones correspondientes en la simulación serial) en las últimas posiciones respecto a la fila de esta matriz. Con el transcurrir del algoritmo esas diferencias se propagan hacia las demás posiciones de la matriz WaterLevelNew. Más adelante se presentan algunas líneas de trabajo tendientes a mitigar estos problemas.

Las matrices que se mencionaron están relacionadas con las ecuaciones gobernantes según se menciona en el reporte técnico de M. Fossati [3] (las ecuaciones 1, 2 y 3 se corresponden con las ecuaciones 55, 56 y 57 de dicho trabajo).

DCoef_2D → *Me%Coef%D2%D*
ECoef_2D → *Me%Coef%D2%E*
FCoef_2D → *Me%Coef%D2%F*
TiCoef_2D → *Me%Coef%D2%Ti*

La forma discretizada de la ecuación de transporte de cantidad de movimiento según *x* se presenta en la ecuación 1, agrupando los términos explícitos en un término común X_{ijk}. Si luego se despejan las velocidades que no se conocen, correspondientes al instante *n+1*, se obtiene la ecuación 2.

$$\frac{U_{1_{ijk}}^{n+1}-U_{1_{ijk}}^{n}}{\Delta t}=\frac{1}{Vu_{ijk}^{n}}\left[\left(\frac{p_{atm_{ij-1}}^{t+1}-p_{atm_{ij}}^{t+1}}{\rho_0}+g\left(\eta_{ij-1}^{n+1/2}-\eta_{ij}^{n+1/2}\right)\right)\cdot Au_{ijk}^{n}+\right.$$
$$\left.+\left(A_{v_{ij-1/2k}}^{n}\frac{U_{1_{ijk+1}}^{n+1}-U_{1_{ijk}}^{n+1}}{DUZ_{ijk}^{n}}-A_{v_{ij-1/2k-1}}^{n}\frac{U_{1_{ijk}}^{n+1}-U_{1_{ijk-1}}^{n+1}}{DUZ_{ijk-1}^{n}}\right)\cdot Ah_{ij-1/2}+X_{ijk}^{n}\right] \qquad \textbf{Ecuación 1}$$

$$\left(-\frac{\Delta t\,Ah_{ij-1/2}}{Vu_{ijk}^{n}}\frac{A_{v_{ij-1/2k-1}}^{n}}{DUZ_{ijk-1}^{n}}\right)U_{1_{ijk-1}}^{n+1}+\left[1+\frac{\Delta t\,Ah_{ij-1/2}}{Vu_{ijk}^{n}}\left(\frac{A_{v_{ij-1/2k-1}}^{n}}{DUZ_{ijk-1}^{n}}+\frac{A_{v_{ij-1/2k}}^{n}}{DUZ_{ijk}^{n}}\right)\right]U_{1_{ijk}}^{n+1}$$
$$+\left(-\frac{\Delta t\,Ah_{ij-1/2}}{Vu_{ijk}^{n}}\frac{A_{v_{ij-1/2k}}^{n}}{DUZ_{ijk}^{n}}\right)U_{1_{ijk+1}}^{n+1}= \qquad \textbf{Ecuación 2}$$
$$U_{1_{ijk}}^{n}+\left[\frac{\Delta t}{Vu_{ijk}^{n}}\left(\left[\frac{p_{atm_{ij-1}}^{t+1}-p_{atm_{ij}}^{t+1}}{\rho_0}+g\left(\eta_{ij-1}^{n+1/2}-\eta_{ij}^{n+1/2}\right)\right]Au_{ijk}^{n}+X_{ijk}^{n}\right)\right]$$

La ecuación 2 es de la forma $Du_{1_{ijk}}\cdot U_{1_{ijk-1}}^{n+1}+Eu_{1_{ijk}}\cdot U_{1_{ijk}}^{n+1}+Fu_{1_{ijk}}\cdot U_{1_{ijk+1}}^{n+1}=TIu_{1_{ijk}}$ con *Du*, *Eu*, *Fu* y *TIu* dados por dicha ecuación. La aplicación de la misma a todos los puntos del dominio de cálculo genera un sistema tridiagonal de ecuaciones que se resuelve eficientemente por el método de Thomas descripto anteriormente. Para la componente *y* de la ecuación de cantidad de movimiento el cálculo es semejante:

$$Du_{2_{ijk}}\cdot U_{2_{ijk-1}}^{n+3/2}+Eu_{2_{ijk}}\cdot U_{2_{ijk}}^{n+3/2}+Fu_{2_{ijk}}\cdot U_{2_{ijk+1}}^{n+3/2}=TIu_{2_{ijk}} \qquad \textbf{Ecuación 3}$$

Siendo:
$$Du_{2_{ijk}}=-\frac{\Delta t\,Ah_{i-1/2j}}{Vv_{ijk}^{n+1/2}}\frac{A_{v_{i-1/2jk-1}}^{n+1/2}}{DVZ_{ijk-1}^{n+1/2}} \qquad Fu_{2_{ijk}}=-\frac{\Delta t\,Ah_{i-1/2j}}{Vv_{ijk}^{n+1/2}}\frac{A_{v_{i-1/2jk}}^{n+1/2}}{DVZ_{ijk}^{n+1/2}}$$
$$Eu_{2_{ijk}}=\left[1+\frac{\Delta t\,Ah_{i-1/2j}}{Vv_{ijk}^{n+1/2}}\left(\frac{A_{v_{i-1/2jk-1}}^{n+1/2}}{DVZ_{ijk-1}^{n+1/2}}+\frac{A_{v_{i-1/2jk}}^{n+1/2}}{DVZ_{ijk}^{n+1/2}}\right)\right]$$

$$TIu_{2_{ijk}} = U_{2_{ijk}}^{n+1/2} + \left[\frac{\Delta t}{Vv_{ijk}^{n+1/2}} \left(\left[\frac{p_{atm_{i-1\,j}}^{n+3/2} - p_{atm_{ij}}^{n+3/2}}{\rho_0} + g\left(\eta_{i-1\,j}^{n+1} - \eta_{ij}^{n+1}\right) \right] Av_{ijk}^{n+1/2} + Y_{ijk}^{n+1/2} \right) \right]$$

3.4.3 Resultados obtenidos respecto al ejemplo

La configuración de subdominios utilizada para las pruebas realizadas con este nuevo enfoque de paralelización del modelo MOHID se muestra en la figura 2. La discretización del dominio original (figura 1) es de 102x101 volúmenes, el tamaño del subdominio 0 es de 51x101 y del subdominio 1 es de 50x101. El ejemplo representa el desarrollo de un flujo unidimensional con un ingreso de caudal en el borde izquierdo y un nivel constante en el borde derecho. El período de simulación es de tres días con un *DT* igual a 10 segundos, esto equivale a realizar 25.920 pasos de simulación. Las variables simuladas son: el nivel del agua (elevación), las corrientes o flujos (formada por dos componentes como la intensidad y la dirección) y la velocidad (formada por las dos componentes Y y X). En la tabla 2 se presenta el tiempo de ejecución (medido en segundos) de la versión serial y de descomposición de dominios.

Versión	Tiempo total de ejecución (s)
Serial	666.35
DDM	853,05

Tabla 2 – Comparación del tiempo de ejecución.

De la tabla 2 se desprende que la versión paralela necesita aproximadamente tres minutos más que la versión serial para completar la simulación, esto se debe a las esperas que se realizan para calcular la matriz resultado dentro de la función THOMAS_2D.

En la tabla 3 se presenta el uso máximo de memoria y el porcentaje de uso respecto a la memoria total de cada nodo que se alcanza en la versión serial y en cada nodo perteneciente a la descomposición de dominios.

Versión	% de uso de memoria	Uso de memoria (Mb)
Serial	3.6	36
DDM	2.1	21

Tabla 3 – Comparación del uso de memoria.

En la tabla 4, 5 y 6 se muestran los valores de las variables simuladas para tres puntos de control, que se escogieron en posiciones distribuidas en el dominio. Estos puntos cumplen que en la descomposición de dominios uno pertenece al dominio de la izquierda y los otros dos puntos pertenecen al dominio de la derecha. Estos puntos se ubican en las coordenadas (36,6) del dominio de la izquierda, (68,8) del dominio de la derecha y (12,13) del dominio de la derecha respectivamente.

Versión	Velocidad X	Velocidad Y	Intensidad	Dirección	Elevación
Serial	-0.9458E-5	0.0000	0.9458E-5	0.2700E+3	-0.9455
DDM	0.2517	0.7912E-3	0.2517	0.8982E+2	0.7529E-2
Err Abs	2.52E-01	7.91E-04	2.52E-1	1.80E+2	9.53E-1
Err Rel	-2.66E+04	N/A	2.66E+4	-6.67E-1	1.01

Tabla 4 – Error relativo y absoluto para el pto (36,6) (dom izquierda).

Versión	Velocidad X	Velocidad Y	Intensidad	Dirección	Elevación
Serial	-0.6346E-4	0.00	0.6346E-4	0.2700E+3	-0.5567
DDM	0.2590	0.4910E-3	0.2590	0.8989E+2	0.3009E-2
Err Abs	2.59E-01	4.91E-04	2.59E-01	1.80E+02	5.60E-01
Err Rel	-4.08E+03	N/A	4.08E+03	6.67E-01	1.01

Tabla 5 – Error relativo y absoluto para el pto (68,8) (dom derecha).

Versión	Velocidad X	Velocidad Y	Intensidad	Dirección	Elevación
Serial	-0.1008E-3	0.00	0.1008E-3	0.2700E+3	-0.4236
DDM	0.2547	0.2923E-3	0.2547	0.8993E+2	0.3405E-2
Err Abs	2.55E-1	2.923E-4	2.546E-1	1.80E+2	4.27E-1
Err Rel	-2.528E+3	N/A	2.526E+3	0.667	-1.01

Tabla 6 - Error relativo y absoluto para el pto (12,13) (dom derecha).

Las figuras 3 a 7 muestran una comparación entre la versión serial y la versión paralela de los valores de las variables *VelocidadX, Intensidad, Elevación, VelocidadY y Dirección* respectivamente.

Se toma en cuenta el punto de control ubicado en las coordenadas (12,13) (dominio de la derecha). Con una tonalidad gris más clara se marca la salida para la versión serial y con negro se marca la salida para la versión paralela. En caso que no haya diferencias en los valores graficados, aparecen solapados. Estos gráficos muestran leves diferencias en la velocidad según y y en consecuencia en la dirección. Es importante relacionar estos resultados con el problema físico que se está representando en este ejercicio de modelación numérica. La componente principal del flujo es según x con valores que convergen a 0.25m/s. La componente y teóricamente debería ser nula pero el modelo calcula pequeños valores que muestran los errores en la aproximación. Estos valores, que no son relevantes, son los que cambian entre la versión serial y la paralela. Los resultados aquí presentados muestran que los valores calculados para las principales variables de este problema, intensidad de flujo, componente según x y elevación de la superficie libre, los resultados obtenidos con la implementación paralela generada son iguales a los obtenidos en la versión serial.

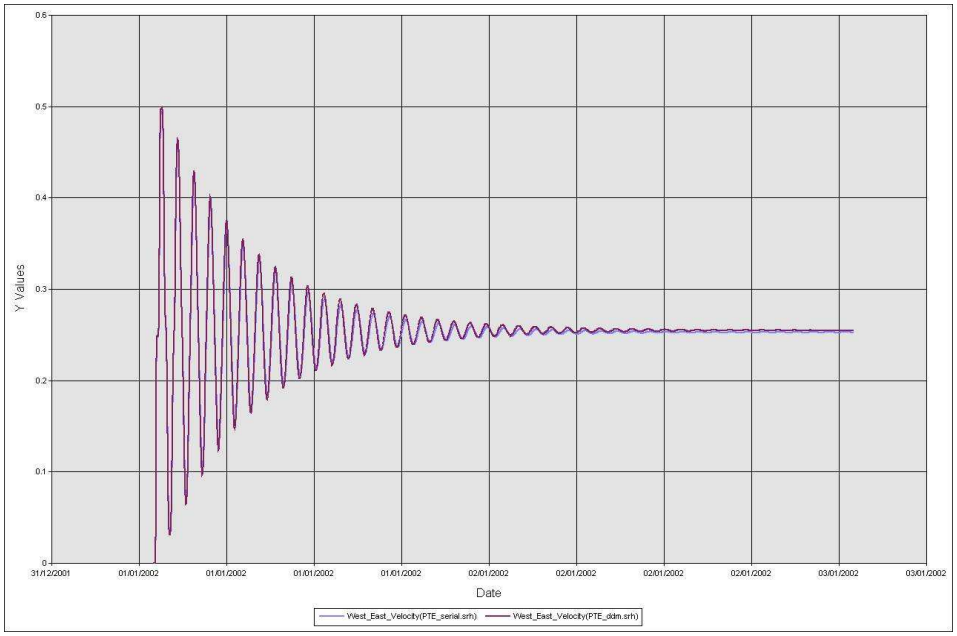


Figura 3 – Comparación de la Velocidad en X.

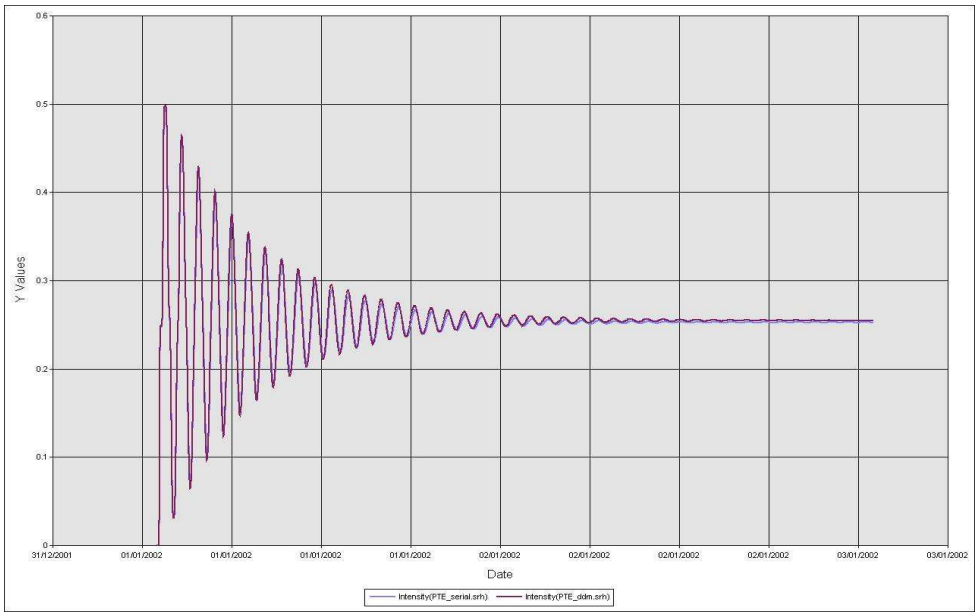


Figura 4 – Comparación de la Intensidad.

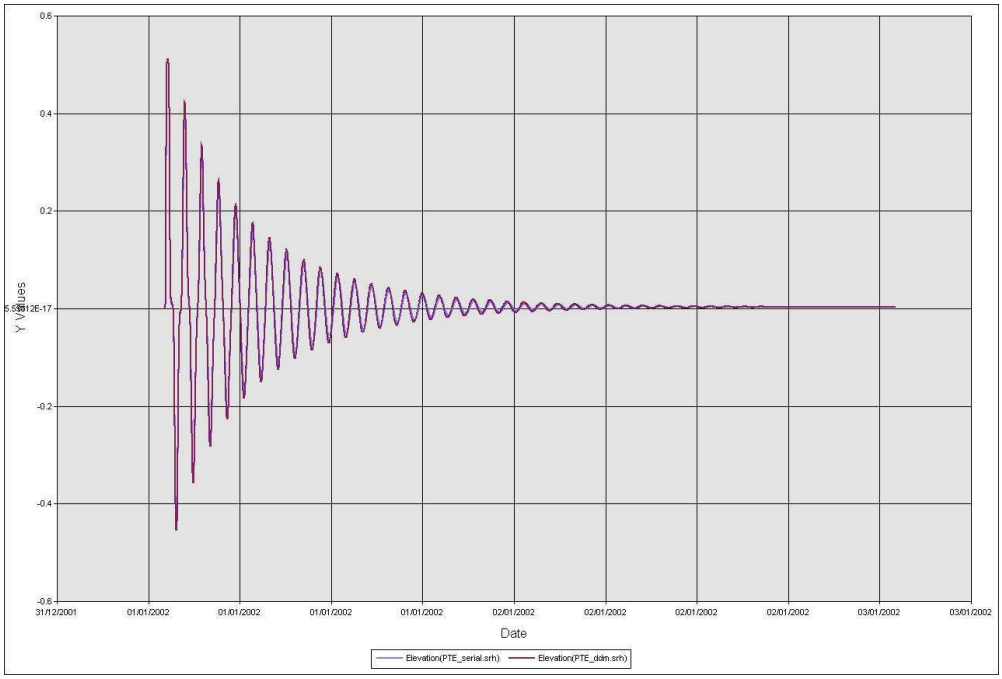


Figura 5 – Comparación de la Elevación.

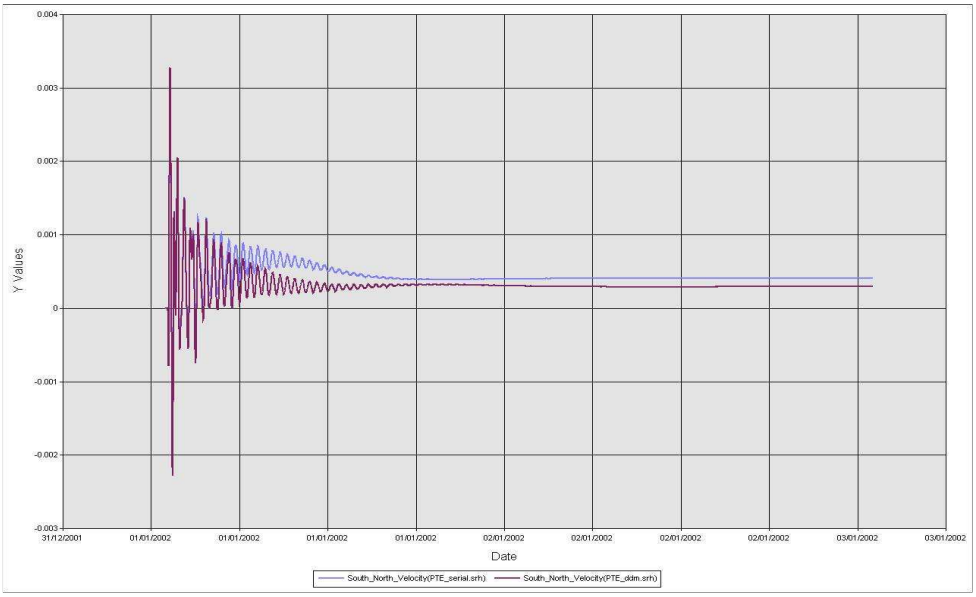


Figura 6 – Comparación de la Velocidad en Y.

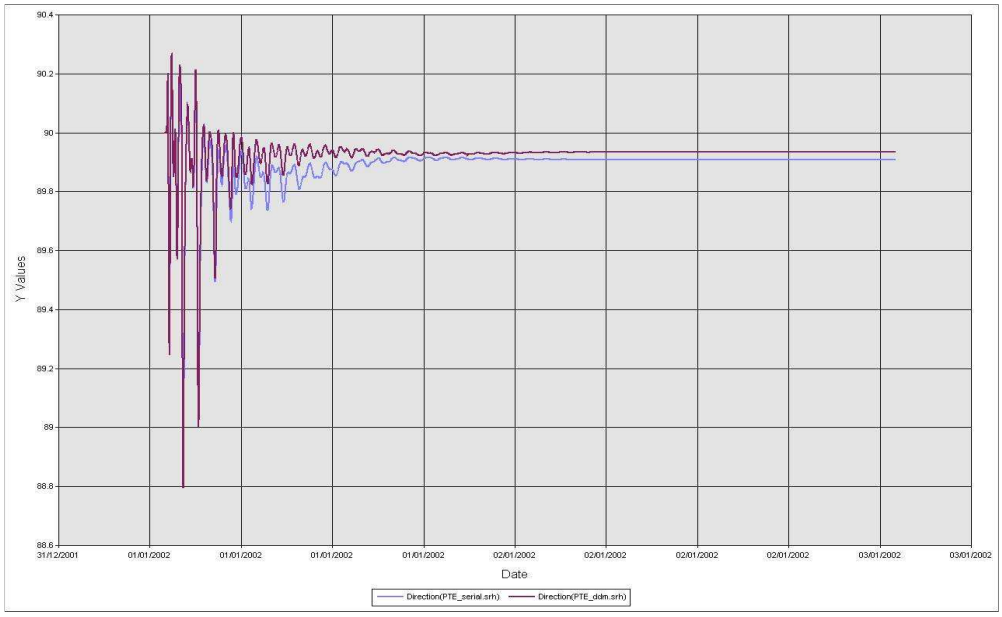


Figura 7 – Comparación de la Dirección.

3.4.4 Extensiones a la versión basada en Thomas

A continuación se comentarán varias líneas de trabajo que quedaron inconclusas y/o parecen interesantes abordar en un futuro cercano.

En primer lugar la versión basada en el algoritmo de Thomas puede ser mejorada en distintos aspectos, ya sea en los resultados numéricos como en la capacidad de ejecutar simulaciones y dominios más complejos. Esta versión se probó para una configuración de dos dominios divididos en forma vertical, esto puede ser naturalmente extendido permitiendo dividir en más dominios. Otra configuración, que es una mezcla de las dos anteriores, es quizás la más genérica y flexible de todas las configuraciones de divisiones que se pueden hacer en forma vertical y horizontal. Para poder llevar a cabo simulaciones con configuraciones de dominios como las mencionadas se deben realizar diversos cambios a nivel de código fuente en las siguientes funciones:

- Compute_WaterLevel
- SendDatosDDMThomas2D
- ReceiveDatosDDMThomas2D
- THOMAS_2D

Los cambios a realizar en el código que se deben hacer en el envío y recepción de datos (son las funciones SendDatosDDMThomas2D y ReceiveDatosDDMThomas2D), se encuentran en ModuleHydrodynamic, en la función Compute_WaterLevel antes del llamado a THOMAS_2D. Por cada dominio se debe enviar los datos a cada dominio vecino y luego éste recibe datos desde cada dominio vecino. Para poder hacer efectivo este cambio, se debe modificar la forma de almacenar los datos que se reciben. Actualmente, como cada dominio tiene solamente un vecino, se reciben datos de un solo vecino. Por cada variable involucrada (en ese caso las variables son DCoef_2D, ECoef_2D, FCoef_2D y TiCoef_2D) para almacenar los datos recibidos es suficiente con un vector por cada variable. Pero en el caso de extender la configuración de

dominios, esto se debe modificar. En vez de almacenar los datos recibidos en un vector por cada variable que sea en una matriz por cada variable. Donde cada matriz representa una variable y cada fila (o columna indistintamente) de esa matriz representa un dominio vecino.

El siguiente cambio a realizar es en la función THOMAS_2D, tanto en el encabezado como en el código, ya que con el cambio propuesto anteriormente se ingresan como parámetro de entrada matrices y no vectores. También se debe recibir la descripción de los vecinos del dominio actual (esto se indica a través de un vector donde cada celda es del tipo T_Vecino). La modificación más importante a realizar es en el algoritmo THOMAS_2D en sí mismo. Observando el código presentado en la sección 4.3.1.3, se deben agregar más condiciones que reflejen todas las posibilidades de ubicación del dominio vecino en las zonas indicadas como “Modificación1”, “Modificación2” y “Modificación3”. Es en esta parte del código, en donde se realiza la comunicación de datos entre los dominios que son vecinos entre sí. En donde el dominio actual sabe a qué vecino mandar datos y de qué vecino esperar recibir datos, para poder avanzar en la resolución del algoritmo.

Por otro lado, se deben agregar más procesos en el archivo de booteo de demonios de MPI, que en este caso se llama mpd.hosts. Debe haber tantas filas como dominios que representan la división del dominio original.

Cada dominio debe tener un archivo llamado “datosVecinos_XX” donde XX es el identificador de proceso de MPI que se asigna en tiempo de ejecución.

4 Conclusiones y trabajo futuro

Como conclusión a destacar es que si bien los diseños realizados son prototipos, con el tercer diseño propuesto (sección 4.3), comparando los gráficos de las figuras 3 y 4 en donde se muestran la evolución de los valores para algunas propiedades del sistema, se puede constatar que este tipo de estrategias pueden ser una opción válida para el escalado en las dimensiones de los modelos a resolver, en particular, cuando el uso de memoria RAM es una de las limitantes.

A continuación se menciona diferentes aspectos a mejorar en el futuro.

Mejorar el pasaje de datos en ModuleWaterProperties con las funciones SendDatosDDMWaterProp y ReceiveDatosDDMWaterProp. Se envían los datos de propiedades del agua, son dos matrices involucradas, corresponden al tipo de datos *T_Property* el cual se encuentra referenciado a través de la variable *Me%FirstProperty*. Este tipo de datos hace referencia a diferentes propiedades del agua, como puede ser la temperatura, salinidad, etc. Las matrices que interesan de esta estructura son *Concentration* y *Assimilation%Field*. Habría que investigar cuál de estas dos matrices, o las dos, precisan estar actualizados sus datos entre los diferentes dominios.

En lo que se refiere a resultados numéricos, incluso en la versión 3 existen diferencias en los resultados obtenidos en la simulación de un modelo ejecutado en forma serial en comparación con la simulación del mismo modelo pero con la versión de descomposición de dominios. Para disminuir estas diferencias se estudió en qué lugar

del código se producen. Estas variaciones ocurren al calcular los volúmenes finitos, dentro de la función llamada ComputeVerticalGeometry dentro de ModuleGeometry (Mohid_Base_2). La diferencia radica en que en el dominio original particionado en la última columna del dominio "dom0" es considerada por MOHID como frontera del dominio, lo cual es cierto (lo mismo ocurre en "dom1" donde la primer columna es considerada como frontera). Sin embargo, en el dominio original esa columna no es de frontera, entonces el modelo realiza diferentes cálculos y asigna diferentes valores cuando una celda es de frontera que cuando no lo es. Es en esa columna donde se producen las diferencias de los valores de las variables simuladas al comparar la versión serial con la de DDM.

Matrices involucradas son:

SZZ → Me%Distances%SZZ
DWZ → Me%Distances%DWZ
DUZ → Me%Distances%DUZ
DVZ → Me%Distances%DVZ

La secuencia de llamadas en donde se modifican estas matrices es la siguiente:
MomentumMassConservation (ModuleHydrodynamic)
    New_Geometry
        ComputeVerticalGeometry (ModuleGeometry)
            ComputeSZZ
            ComputeDistances
            ComputeVolumes
            ComputeAreas
            ComputeZCellCenter
            ComputeWaterColumn

Se creó la versión v9, no detallado en este reporte, con esa sección modificada, pero los resultados no fueron completamente satisfactorios. Es necesario profundizar en el estudio de la repercusión de los valores que se allí se calculan.

Un trabajo a realizar es la investigación en cómo aplicar la técnica de Descomposición de Dominios, ya sea a través del método multiplicativo, aditivo, de complemento de Schur u otra técnica similar dentro de la familia de DDM, en el sentido de resolver la frontera entre los dominios involucrados y después hacer los cálculos para los valores interiores en el dominio, o el paso inverso, calcular en forma interna y luego hacer cálculos en la frontera.

Referencias

[1] I. Barreto, P. Ezzatti, M. Fossati. Estudio inicial del modelo MOHID. Reporte técnico RT 09-10. PEDECIBA Informática, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, mayo de 2009.

[2] G. Golub, C.V. Loan. Matrix Computations. The Johns Hopkins University Press, 1996.

[3] M. Fossati. Modelación numérica de las ecuaciones de Navier-Stokes. Trabajo final. Análisis numérico del modelo tridimensional en volúmenes finitos MOHID. Julio 2006.

[4] I. Barreto, P. Ezzatti, M. Fossati. Instalación de MOHID en Linux. Reporte técnico RT 10-03. PEDECIBA Informática, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, mazo de 2010.

[5] Sitio web de la ayuda on-line del Mohid. http://www.mohid.com/wiki/index.php?title=Nix_platforms. Última consulta Noviembre 2010.

[6] Stitio web del grupo de desarrlo de HDF5. http://www.hdfgroup.org/HDF5/. Última consulta Febrero 2011.

[7] A. Toselli , O. Widlund. Domain Decomposition Methods - Algorithms and Theory. Springer Series in Computational Mathematics, Vol. 34, 2005.

Anexo A – ModuleHydrodynamic, ActualizeSubModelValues

A continuación se presenta el código simplificado de la función ActualizeSubModelValues que se encuentra en ModuleHydrodynamic. Se hace énfasis en los cálculos sobre las matrices de interés, es decir los datos que se obtienen del dominio padre y se actualizan en el submodelo hijo.

ACTUALIZE SUB MODEL VALUES

call **GetGridData** (obtiene un puntero a Bathymetry)

call **GetGeometryMinWaterColumn** (obtiene un puntero a MinWaterColumn)

call **ReadLockSon** (obtiene punteros a ImposedTangFacesUSon,
ImposedTangFacesVSon,
ImposedNormFacesUSon,
ImposedNormFacesVSon,
Water3DSon,
Boundary2DSon,
Faces3D_USon,
Faces3D_VSon,
DUZ_Son,
DVZ_Son,
DXX_Son,
DYY_Son)

(La primera vez)
---- si se cumplen ciertas condiciones ----

Me%WaterFluxes%X(i, j, k) = Me%SubModel%qX (i, j, k) * DYY_Son(i, j)

Me%WaterFluxes%Y(i, j, k) = Me%SubModel%qY (i, j, k) * DXX_Son(i, j)
(fin si primera vez)

(en el caso de water3DSon)
Me%SubModel%Z(i, j) = Me%SubModel%Z_Next (i, j) * TimeCoef +
Me%SubModel%Z_Previous(i, j) * (1 - TimeCoef)

Me%SubModel%Z(i, j) = - Bathymetry(i, j) + 0.75 * MinWaterColumn

Me%WaterLevel%New(i, j) = Me%SubModel%Z(i, j)

call **RemoveLowerSpikes** (Me%WaterLevel%New, Water3DSon, Bathymetry,
MinWaterColumn, Me%SubModel%DeadZonePoint,
Me%SubModel%DeadZone, ILB, IUB, JLB, JUB, KUB)

(en el caso de boundary2DSon)
Me%SubModel%Z(i, j) = Me%SubModel%Z_Next (i, j) * TimeCoef +
Me%SubModel%Z_Previous(i, j) * (1 - TimeCoef)

Me%SubModel%Z(i, j) = - Bathymetry(i, j) + 0.75 * MinWaterColumn

(si Faces3D_USon == Covered) (lo mismo con Faces3D_Vson, se cambia la U por V en las matrices de la estructura Me%SubModel)

```
Me%SubModel%U_New(i, j, k)  =  Me%SubModel%U_Next    (i, j, k) * TimeCoef  +
                    Me%SubModel%U_Previous(i, j, k) * (1 - TimeCoef)


Me%Velocity%Horizontal%U%New(i, j, k) = Me%SubModel%U_New(i, j, k)


Me%SubModel%DUZ_New(i, j, k) = Me%SubModel%DUZ_Next (i, j, k) * TimeCoef+
                    Me%SubModel%DUZ_Previous(i, j, k) * (1 - TimeCoef)
```

(si ImposedTangFacesUSon == Imposed) (lo mismo para V)

```
Me%SubModel%U_New(i, j, k)  = Me%SubModel%U_Next    (i, j, k) * TimeCoef  +
                                  Me%SubModel%U_Previous(i, j, k) * (1 - TimeCoef)
```

(luego corrige con 0 en ciertos puntos)
(si es la primera vez y es continua) (lo mismo para V)

```
Me%Velocity%Horizontal%U%New(i, j, k) = Me%SubModel%U_New(i, j, k)

Me%SubModel%DUZ_New(i, j, k) = Me%SubModel%DUZ_Next (i, j, k) * TimeCoef  +
                        Me%SubModel%DUZ_Previous(i, j, k) * (1 - TimeCoef)
```

Anexo B – Llamadas dentro ReadNextOrInitialField y ActualizeSubModelValues

En este Anexo, se detalla las llamadas que se hacen dentro de la función ReadNextOrInitialField y la invocación a ActualizeSubModelValues que se encuentran dentro del procedimiento SetWaterPropFather en ModuleWaterProperties. En ModuleHydrodynamic también se encuentra la función ReadNextOrInitialField que hace las mismas llamadas (InterpolRegularGrid) modificando algunos parámetros en la invocación.

```
call ReadNextOrInitialField(
                        FatherGridID = FatherGridID,                    &
                        Open3DFather = Open3DFather,                    &
                        FatherZCellCenter = FatherZCellCenter,          &
                        PropertySon  = PropertySon,                     &
                        PropFatherConcentration = PropFatherConcentration,  &
                        InitialField = InitialField)

    call GetComputeZUV(FatherGridID, ComputeZ, STAT = STAT_CALL)

    if (PropertySon%SubModel%InterPolTime .and. .not. InitialField)then
      call SetMatrixValue(PropertySon%SubModel%PreviousField,       &
                  Me%Size, PropertySon%SubModel%NextField)
    endif
    !Ang: new implementation
    if ((PropertySon%SubModel%VertComunic == FatherSonDifDim) .or.     &
      (PropertySon%SubModel%VertComunic == Father3DSon2D)) then

      call InterpolRegularGrid   (Me%ObjHorizontalGrid,             &
                        FatherGridID,                   &
                        PropFatherConcentration,        &
                        PropertySon%SubModel%Aux_Field,     &
                        Open3DFather, ComputeZ,         &
                        KLBFather, KUBFather, KUBSon,       &
                        STAT = STAT_CALL)

      call InterpolRegularGrid   (Me%ObjHorizontalGrid,             &
                        FatherGridID,                   &
                        FatherZCellCenter,              &
                        PropertySon%SubModel%Aux_ZCellCenter, &
                        Open3DFather, ComputeZ,         &
                        KLBFather, KUBFather, KUBSon,       &
                        STAT = STAT_CALL)
    else

      call InterpolRegularGrid   (Me%ObjHorizontalGrid,             &
                        FatherGridID,                   &
                        PropFatherConcentration,        &
                        PropertySon%SubModel%NextField,     &
                        Open3DFather, ComputeZ,         &
                        KLBFather, KUBFather, KUBSon,       &
                        STAT = STAT_CALL)
    endif
```

```
      if (PropertySon%SubModel%InterPoLTime .and. InitialField)then

         if ((PropertySon%SubModel%VertComunic /= FatherSonDifDim) .or.  &
            (PropertySon%SubModel%VertComunic /= Father3DSon2D)) then

            call SetMatrixValue(PropertySon%SubModel%PreviousField,     &
                        Me%Size, PropertySon%SubModel%NextField)
         endif

      end if
```

call **ActualizeSubModelValues** (PropertySon, PropFatherOld, InitialField)

```
      PropertySon%Assimilation%Field(i, j, k) =  PropertySon%SubModel%NextField (i, j, k) *
TimeCoef  +
                  PropertySon%SubModel%PreviousField(i, j, k) * (1 - TimeCoef)
      else
            PropertySon%Assimilation%Field(i, j, k) = PropertySon%Concentration(i, j, k)
```