# Meta volante #3 - Llegando a las estrellas con PRESTO

Vincent A. Arcila*, Jhon Jairo Chavarría Gaviria†, Sebastián Obando‡, Juan M. Young Hoyos§

*Universidad EAFIT* *†§
*Institución Universitaria EAM* ‡
*Colombia*

**vaarcilal@eafit.edu.co,* †*jjchavarrg@eafit.edu.co,* ‡*sebastian.obando.8888@eam.edu.co,* §*jmyoungh.edu.co*

***Abstract*—En el presente documento se describe el proceso de de implementación para la solución de la meta volante #3. Este documento sirve como referencia para entender la experiencia del equipo configurando y trabajando con PRESTO.**

## 1. Entorno del cluster

### 1.1. Hardware

- **Cantidad de nodos:** 3×ProLiant SL230s Gen8.
- **Procesadores por nodo:** 3×CPU Intel® Xeon® CPU. E5-2670, 8 núcleos por procesador, hyper-threading desactivado.
- **Controlador Infiniband:** Mellanox Technologies MT27500 Family ConnectX-3
- **Memoria por nodo:** 16 x 4GB DIMM DDR3 1333 M3T/s.

### 1.2. Software

- **Intel OneAPI ?**.
- **HPL 2.3 Intel version ?**.
- **Operating system:** Centos 8.2 **?**.
- **BLAS:** BLAS en MKL **?**.
- **MPI:** Intel MPI **?**.
- **Compilador:** Intel C Compiler (envuelto por Intel MPI) **?**.
- **NFSv4 ?**.
- **Manejador de recursos**: Slurm 20.11.8 **?**.
- **Drivers de Infiniband**: Mellanox OFED 4.9 LTS.

## 2. Configuración del Cluster

Para empezar con la configuración del cluster iniciamos instalando unos paquetes básicos, esto usando los archivos de *Ansible* que teníamos, y con esto se instalaron paquetes básicos como editores de texto (*Vim* o *Emacs*), drivers (como por ejemplo los de *Infiniband*) y *ntp*, además se instalaron los compiladores de *Intel*, *NFS*, *NFSoRDMA* (aunque se estudiaba la posibilidad de usar *GlutterFS*), también *SLURM*, *QUEST* y por último *PRESTO*, ahora bien, para poder instalar *PRESTO* primero debiamos tener algunas dependencias instaladas como *glib* (versión 2.66.4), *fftw* (versión 3.3.9), *cfitsio* (versión 3.49), *tempo* (versión 1.0.0) y *pgplot* (versión 5.2).

## 3. Estudio de performance

El máximo performance teórico de nuestro cluster está determinado por la siguiente fórmula.

$$R_{peak} = \#nodos \times \frac{\#cpus}{nodo} \times \frac{\#nucleos}{cpu} \times \frac{ciclos}{segundo} \times \frac{FLOPs}{ciclo} \quad (1)$$

Para nuestro sistema, se tenia el siguiente $R_{peak}$:

$$R_{peak} = 3 \times 2 \times 8 \times 2.6 \times 10^9 \times 8 = 998.4\ GFLOP/s \quad (2)$$

## 4. El camino a ser los mejores

1) **Backups**. Diarios, para que no se pierda nada importante.
2) **Montar NFS**.
3) **Instalar drivers de Mellanox**.
4) **Instalar SLURM**. Esto nos sirvió para coordinar nuestras ejecuciones cuando se trabajo individualmente.
5) **Instalar Intel**. Se instalo Intel OneAPI **?** Base y HPC toolkits, que traían consigo Intel MPI **?** e Intel MKL **?**.
6) **Resultados base**. Se ejecuto HPL usando MKL e Intel MPI usando la versión base de HPL. Estos resultados nos sirvieron para saber nuestro punto de partida cuando se uso el HPL de Intel.
7) **Optimización con Intel HPL**. Es la fase donde empezamos a ejecutar autotuners en masa que nos pudieran mostrar los valores óptimos de la configuracion de HPL.

### 4.1. Backups

Al momento de trabajar con un cluster que tiene bastante tiempo de funcionamiento es posible perder el trabajo realizado, ya que puede fallar definitivamente, estropenado la información del disco **?**. Para evitar que se arruine el progreso obtenido es recomendable hacer Backups frecuentemente. Nosotros decidimos hacerlos todos los días. Por lo tanto, estas copias se harán en el servidor `cronos.eafit.edu.co` reservadas para el usuario `proxyjump`. **?**.

Para el trabajo se hicieron las sincronizaciones con Rsync. Esta es una herramienta para realizar sincronización de archivos remotos y locales, usando un algoritmo que solo copia los archivos que cambiaron, optimizando el uso de recursos en la máquina. Además, para ejecutar este trabajo automáticamente se programó un archivo bash que fue corrido desde *Crontab*, el cual permite hacer tareas con la frecuencia que se elija mediante ciertos comandos **?**.

### 4.2. Montar NFS

Un *sistema de archivos de red (NFS por su siglas en inglés)* permite acceder a un sistema de archivos remotos a través de la red, facilitando la administración de recursos en servidores centralizados **?**. Este sistema es requerido por MPI para su correcto funcionamiento. NFS funciona bastante bien para un número reducido de nodos, pero hay mejores opciones como GlusterFS o Lustre **?**. Nuestro servidor NFS estuvo montado en el `compute-1-1` y el cliente en `compute-1-2`.

### 4.3. Infiniband

Infiniband es una conexión para acelerar la comunicación entre nodos. En la instalación de Infiniband se usó el driver Mellanox. Se instaló la versión SRC, de modo que nos permitiera compilar todos los paquetes para nuestra versión se kernel específica. Se activó en tiempo de instalación RDMA, NFSoRDMA e IPoIB. Se activó también la instalación de todos los paquetes que vienen con los drivers.

### 4.4. Intel

Se instaló Intel porque puede compilar aplicaciones para que funcionen mejor con el hardware y además trae una versión optimizada de HPL. Instalar Intel es fácil, solo se descarga y se corre el bash para construirlo. Más adelante se mostrarán los flags de compilación y variables de ambiente para mejorar la compilación y ejecución de HPL con Intel.**?**

### 4.5. Paquetes

Para instalar estas aplicaciones y otras se tuvieron que incluir múltiples paquetes. Los paquetes que se instalaron fueron usados para poder instalar los drivers y las aplicaciones. Además se instalaron varios paquetes para optimizar la comunicación de los nodos. Todos los paquetes instalados están en el documento Paquetes.txt.zip.

### 4.6. Slurm

Slurm es un sistema de control para supercomputadores. Permite controlar los procesos de manera que corran con todos los recursos que necesitan. Slurm hay que instalarlo en los dos nodos. La instalación de Slurm se debe hacer después de instalar NFS. Además de esto hay que instalar `munge` para usar su servicio de autenticación. Para compilar Slurm se usó `march-native`, que optimiza la ejecución de Slurm.

Se configuró Slurm desactivando el soporte para `cgroup`, para estar seguros que no se restringían los recursos en las ejecuciones.

### 4.7. Montar NTP

NTP es un protocolo usado para sincronizar los relojes de las computadoras dentro de una red. Para ello se usará la aplicación Chrony, la cual se instaló usando el manejador de paquetes. De esta forma, se sincronizó `compute-1-1` a la hora oficial de Bogotá usando el servidor `3.co.pool.ntp.org`. Después, se usó `compute-1-1` como servidor para `compute-1-2`.

## 5. La mera optimización ganadora

En esta sección se expandió sobre las técnicas ganadoras. Se intentó ver a HPL desde todos los ángulos posibles, intentando aumentar FLOPs con todas las herramientas que se tuvo.

### 5.1. Archivo de configuración

- **N**: Este es parámetro que más impacta el resultado. Define el tamaño de la matriz. El objetivo debe ser siempre elegir el valor más grande que quepa en RAM para maximizar la proporción de computación efectiva **?**, sin causar que el sistema haga swapping. Hay que tener en cuenta que el sistema operativo ocupa espacio en memoria, por lo que la matriz no debe ocupar el 100% de la RAM. El porcentaje óptimo de memoria a ocupar está alrededor del 80% **?**. Existe una fórmula para hallar el N que representa el tamaño de la memoria:

  $$\sqrt{(TotalMem \times (1024^3) \times Nodos)/8}$$

  Donde *TotalMem* es el tamaño de la memoria en GB y *Nodos* es el número de nodos. También se pued multiplicar por el factor que representa el porcentaje de memoria que se quisiera usar, por ejemplo, para el 90% se tiene:

  $$\sqrt{(TotalMem \times (1024^3) \times Nodos)/8} \times 0.90$$

  En nuestros nodos, había alrededor de 61.3 GiB libres en cada uno. No se puede utilizar completamente esta cifra, pues los buffers de memoria y MPI necesitan espacio.

- **NB**: Define el tamaño del bloque para la distribución de datos. En general son menores a 256 y los mejores valores suelen ser múltiplos de 32 **?**. Además, se consideró que lo mejor es que N fuera divisible entre NB, para hacer un uso más eficiente de la memoria. Para este trabajo se estableció un valor de 256, ya que en varias fuentes se evidenció que era el mejor para nuestro procesador.
- **P**×**Q**: El producto de **P** y **Q** es la cantidad de procesos que HPL va a utilizar. **Q** debe ser un poco mayor de **P**.
  Para la versión de Intel HPL, la cantidad óptima de procesos es la cantidad de sockets que se tienen.
- Panel Factorization: Hay tres opciones para factorizar el panel: left, crout, y right. Depende de la máquina de cual opción es la mejor. Para nuestros nodos la opción mas rápida fue right. Esto se puede modificar cambiando los PFACs.
- Recursion Factorization: Hay tres opciones para factorizar la recursión: left, crout, y right. Depende de la maquina cual es la mejor opción. Para nuestros nodos la opción más rápida fue right. Esto se puede modificar cambiando los RFACs.
- Panel Broadcast: Las mejores configuraciones de panel broadcast son increasing ring modified 1 y 2. Esto es porque mandan la información mucho más rápido al segundo nodo haciendo que toda la comunicación se mueva mas rápido.
- Look ahead: Este parámetro hace que el programa mande información al siguiente panel. Esto lo cambias modificando el parametro de depth. Puede mejorar la velocidad de procesamiento pero hay que tener cuidado porque toma mucha memoria.
- Update: Hay dos formas de update binary-exchange y long. Long normalmente es mejor para problemas mas grandes. Esto lo puedes cambiar cambiando el número del SWAP. Además se puso el threshold swapping en 128.

## 5.2. Flags de compilación

Para optimizar la compilación de HPL de Intel se usaron varias flags de compilación.

- -Ofast: Es una flag que incluye varias otras flags que son -O3, -no-prec-div, y -fp-model fast=2. -O3 optimiza el programa para algoritmos que tienen complejidad $O(n^3)$. -no-prec-div le quita precisión a las operaciones de punto flotante. -fp-model fast=2 hace que el programa corra lo mas rápido posible.
- -xHost: Le dice al compilador que produzca instrucciones para el host.**?**
- -ansi-alias: Pone las reglas de ANSI en effecto.
- -fdefer-pop: Quita las funciones justo cuando retornan sus valores.
- -foptimize-sibling-calls: Optimiza las funciones de recursión de cola.
- -mbranches-within-32B-boundaries: Fusiona las ramas con limites 32-byte para mejorar rendimiento.
- -ip: Determina si están habilitadas las optimizaciones entre procedimientos adicionales para la compilación de un solo archivo.
- -qoverride-limits: Deja sobreponer sobre unos limites del compilador. Previene uso excesivo de la memoria.
- -use-intel-optimized-headers: Determina si el directorio de encabezados de rendimiento se agrega a la lista de búsqueda de ruta.
- -pcn 64: Pone la precisión de los número flotante en doble.
- -mkl=parallel: Importa la librería de paralelo del Math Kernel Library.
- -Wall: Pone diagnósticos de warning y error.
- -fPIC: Determina si el compilador código es de posición independiente.

## 5.3. Variables de ambiente de MPI

Se utilizó el protocolo de datagramas de usuario (UDP) para realizar el intercambio de mensajes de MPI debido a que es una alternativa mas escalable que consume menos memoria, este se activó utilizando los comand:**?**

- export I_MPI_FABRICS=shm:dapl
- export I_MPI_DAPL_UD=1

Para que se haga la transferencia utilizando RDMA se utilizó el comando:

- export I_MPI_DAPL_UD_PROVIDER=ofa-v2-ib0

Se deshabilitó la opción de utilizar una fábrica de comunicación distinta de DAPL en caso de no poder inicializarla mediante el comando:

- export I_MPI_FALLBACK=0

Se quería ver la cantidad de datos enviada por cada proceso, así como información de tiempos en las funciones de MPI, esto se realizo usando:

- export I_MPI_STATS=1-20

Para guardar esa información se eligió una carpeta usando

- export I_MPI_STATS_FILE=<path de la carpeta>

Se estableció el número de procesos por nodo en 4 utilizando:

- export I_MPI_PERHOST=4

Para evitar que el agendador de MPI establezca el número de procesos por nodo físico y que sea el que se estableció se debe deshabilitar el asignador de procesos usando:

- export I_MPI_JOB_RESPECT_PROCESS_ PLACEMENT=0

También se realizaron optimizaciones sobre los algoritmos utilizados para realizar ciertas operaciones de MPI que se utilizan en HPL. Estas fueron:

- **Allgather y allgatherv:**
  Para realizar esta operación el algoritmo ring

resulta ser mas eficiente para mensajes de más de 512K y cuando se transmiten mensajes de menor longitud el algoritmo recursive doubling es mejor. Para establecer estos algoritmos se utilizan los comandos:

  - export I_MPI_ADJUST_ALLGATHER="1:0-524288;3"
  - export I_MPI_ADJUST_ALLGATHERV="1:0-524288;3"

- **Reduce:**
  Para operaciones con datos de menos de 2Kb el algoritmo binomial resulta más eficiente mientras que para operaciones con mas tamaño de datos es más eficiente el algoritmo de Rabenseifner, esto se estableció con el comando:

  - export I_MPI_ADJUST_REDUCE="2:0-2048;5"

- **Allreduce:**
  El caso de la operación allreduce es similar al de reduce, sin embargo el algoritmo más eficiente con menos de 2Kb es el algoritmo recursive doubling, se estableció con:

  - export I_MPI_ADJUST_ALLREDUCE="1:0-2048;2"

- **Broadcast:**
  En el caso del broadcast el algoritmo que nos da un mejor performance con mensajes de menos de 12 Kb resulta ser el binomial, se estableció con:

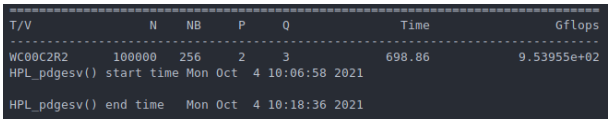
  - export I_MPI_ADJUST_BCAST="1:1-12288"

# 6. High Performance Linpack Benchmark

Usamos el benchmark HPL incluido en la biblioteca OneAPI; este punto de referencia está disponible en *oneapi/mkl/2021.4.0/benchmarks/mp_linpack/*. El directorio mencionado anteriormente incluye: (1) *xhpl_intel64_dynamic*: un binario de referencia HPL listo para usar; (2) *runme_intel64_dynamic*: un script de shell donde se establecen las variables de entorno MPI, aquí el binario de referencia HPL se llama con la biblioteca MPI; (3) *HPL.dat*: el archivo de parámetros de HPL.

# 7. Resultado sorprendente

Después de todo este trabajo de optimización se pudo obtener un resultado 953.95 GFLOPs, equivalente a una eficiencia de 95.54%.




# 8. Quantum Exact Simulation Toolkit (QuEST)

QuEST es un simulador de circuito cuántico que representa el estado del sistema cuántico utilizando vectores y matrices de densidad.

La descripción del estado de un sistema cuántico de $n$ qubits requiere $2^n$ números complejos. Por lo tanto, se cree que la complejidad subyacente en la simulación de computadoras cuánticas utilizando máquinas clásicas es exponencial con respecto al número de qubits.**?**.

## 8.1. Informe de optimizaciones del compilador

Para conocer las optimizaciones automáticas realizadas por el compilador en el código fuente de QuEST, se colocó la marca de informe de optimización del rendimiento, *-qopt-report = 5*, además de las otras marcas del compilador Intel C. Para esta utilidad de informe se consideró el nivel de informe más alto (cinco). Esta utilidad de informes genera varios archivos *.optrpt*, uno por archivo *.cpp* en QuEST.**?**

## 8.2. Optimizaciones de rendimiento

Ejecutamos los problemas ronda preliminar random.c, GHZ QFT y el problema 4 usando solo procesos MPI Distribuido. El rendimiento entregado en la configuración distribuida se utilizó como línea base para el cálculo de la aceleración y la validación de cada mejora de rendimiento. La primera mejora de rendimiento aplicada fue la adición de Multi-threading, que se aplicó agregando soporte para multi-threading en el archivo QuEST makefile de cada problema.

## 8.3. Variables de ambiente de MPI y SLURM

- export OMP_NUM_THREADS=16
- export I_MPI_CC=icx
- export I_MPI_PMI_LIBRARY=/opt/slurm/20.11.5/lib/libpmi2.so
- export SLURM_CPU_BIND="quiet,boards"
- export SLURM_PMI_KVS_NO_DUP_KEYS=1
- export SLURM_MPI_TYPE=pmi2
- export CC=icx
- export CXX=icpx
- export FC=ifort
- export F90=ifort
- export F77=ifort
- export MPIF90=mpiifort
- export LD=ld
- export AR=ar
- export UCX_NET_DEVICES=mlx4_0:1