Meta volante #0 - La mejor metodología para el peor resultado

Vincent A. Arcila*, Santiago Alzate†

Departamento de Informática y Sistemas, Universidad EAFIT
Medellín, Colombia

vaarcilal@eafit.edu.co, †salzatec1@eafit.edu.co

***Abstract*—El presente documento tiene por objetivo demostrar los procesos necesarios para poder ejecutar HPL en nuestro cluster de 2 nodos en Cronos. Este documento servirá como ejemplo de documentación, demostrando la calidad mínima que se espera en las entregas de las siguientes metas volantes. Se ignoraron oportunidades de optimización en aras de que cada equipo pueda explorar las optimizaciones por su cuenta.**

1. Entorno de ejecución

El entorno de ejecución tiene bastante relevancia en el performance de las aplicaciones que se ejecutan sobre el sistema; tanto la infraestructura física como el software base tienen implicaciones sobre los límites de performance.

1.1. Hardware

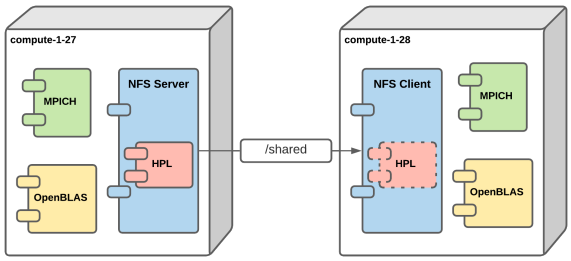
- **Cantidad de nodos:** 2.
- **Procesadores por nodo:** 2×CPU Intel® Xeon® CPU. E5-2670, 8 núcleos por procesador, hyper-threading desactivado.
- **Controlador Ethernet:** Intel Corporation I350 Gigabit Network Connection
- **Memoria por nodo:** 16 x 4GB DIMM DDR3 1333 MT/s.

1.2. Software y dependencias

- **HPL 2.3** [1].
- **Operating system:** Centos 8.2 [2].
- **BLAS:** OpenBLAS 0.3.12 [3].
- **MPI:** MPICH 3.3.2 [4].
- **Compilador:** GCC 8.4.1 [5].
- **NFSv3** [6].

Instalamos OpenBLAS y MPICH usando el manejador de paquetes por defecto. De haber intentado optimizar a nivel de compilación, habríamos instalado todas las dependencias manualmente usando el compilador de Intel, activando flags especiales para generar código optimizado para la aquitectura específica del procesador como xHost [7].

Figure 1. Nuestra infraestructura - Diagrama de despliegue



Como solo necesitábamos ejecutar HPL una sola vez, decidimos no instalar Slurm. Si hubiéramos querido optimizar, habríamos podido diseñar ejecuciones en masa de HPL usando Slurm para encontrar la mejor combinación de parámetros en `HPL.dat`. Esta técnica de optimización se llama auto-tuning.

Tampoco usamos el compilador de Intel para HPL, y decidimos usar Ethernet en lugar de Infiniband, como decisión estratégica en nuestra campaña de ser los peores. Por otro lado, NFS fue instalado con la versión 3 en lugar de la versión 4, lo que merma el performance [8]. Sin embargo, al no ser HPL una aplicación intensiva en escritura, no es muy relevante el performance del sistema de archivos compartido.

2. El Heaven de HPL

En la optimización de software, el Heaven es el mejor performance alcanzable en la máquina que se está usando. Semejante a en tu vida, siempre quieres alcanzar el Heaven. El Heaven de HPL se conoce como R_{peak}, y es la máxima cantidad de operaciones de punto flotante que teoréticamente el sistema es capaz de computar.

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

R_{peak} toma en cuenta todas la unidades de cómputo que se están usando. Es por esto que en la fórmula se tienen en cuenta todos los núcleos de todas las CPUs de todos los

nodos. En arquitecturas de CPU recientes, el mejor valor de FLOPs por ciclo que se puede tener va a ser determinado por el ancho de la unidad de SIMD de la CPU. Cuando la máquina soporta instrucciones Fused Multiply-Add (FMA) se puede hacer una multiplicación y una suma en un solo ciclo de reloj, mejorando los FLOPs. Normalmente cuando se usan instrucciones SIMD de última generación (p. ej. AVX512 [9]) la frecuencia del reloj de la CPU va a ser más baja, pero aún así dando más FLOPs por segundo que sin instrucciones SIMD.

R_{peak} es un valor teórico que se usó para medir la eficiencia de HPL, y en la realidad no hay configuración que alcance R_{peak}. Es un valor de referencia del que se puede obtener una meta final. Los valores normales de eficiencia de HPL en supercomputadores reales del Top500 [10] están entre 50% y 80% [11]. Cuando se llega a una eficiencia relativa aceptable a comparación de R_{peak} se termina de optimizar HPL.

En nuestros nodos el mejor set de instrucciones para operaciones vectoriales es AVX [9]. Con precisión doble, AVX puede hacer 8 operaciones de punto flotante por ciclo [12].

Para nuestro sistema, tenemos el siguiente R_{peak}:

$$R_{peak} = 2 \times 2 \times 8 \times 2.6 \times 10^9 \times 8 = 665.6\ GFLOP/s \quad (2)$$

3. El camino a HPL

Los pasos necesarios para tener HPL corriendo en dos nodos fueron:

1) **Montar NFS**.
2) **Instalar MPI**. HPL necesita una implementación de MPI [13] para ejecutarse en distribuido.
3) **Instalar BLAS**. HPL soluciona un sistema de ecuaciones y usa la interfaz de BLAS [14] para algunas de sus rutinas.
4) **Instalar y ejecutar HPL**.

3.1. Montar NFS

Un sistema de directorios compartido es necesario para el funcionamiento de MPI. Existen soluciones obscuras que aseguran poder usar MPI sin un directorio compartido [15], pero montar un sistema NFS es lo suficientemente fácil para no intentar métodos extraños de un blog random de internet.

NFS es una solución muy común para sistemas de pocos nodos donde se necesita sincronizar almacenamiento por red. El performance de NFS es bastante aceptable, sin embargo hay otras soluciones de almacenamiento compartido que escalan mejor cuando se aumentan la cantidad de nodos como GlusterFS [16] o Lustre [17].

Dado que HPL solamente escribe a disco cuando resume los resultados en un archivo de texto, las optimizaciones que se puedan hacer en este frente no impactan el performance.

Como se puede ver en la figura 1, nuestro servidor NFS estuvo en `compute-1-27` y nuestro cliente en `compute-1-28`.

3.2. Instalar MPI

Message Passing Interface (MPI) es un estándar que define funciones de una biblioteca diseñada para computación distribuida. Resolviendo el problema de la comunicación, MPI permite que una aplicación cree varios procesos que trabajen sincronizados para la solución de un problema.

Creada en los 90's, MPI se ha convertido en una biblioteca utilizada ampliamente en aplicaciones paralelas por su portabilidad y su performance.

Hay muchas implementaciones de MPI y cualquiera puede ser utilizada para HPL. Cada implementación tiene sus puntos fuertes, y la elección de cuál es la mejor parece ser una pregunta que cambia de respuesta con cada aplicación. Algunas de las implementaciones de MPI más relevantes son OpenMPI [18], MPICH [4], MVAPICH2 [19] e Intel MPI [20].

Nosotros elegimos MPICH para nuestra ejecución de HPL. Instalamos MPICH en cada nodo usando el manejador de paquetes del sistema operativo. No configuramos MPICH de ninguna manera. Si hubiéramos querido optimizar MPICH se habría podido compilar usando el mejor set instrucciones de la máquina, y configurando la instalación para usar Infiniband.

Generalmente las implementaciones de MPI traen consigo su propio compilador. Este compilador es un envoltorio de algún otro compilador (p. ej. de GCC), con funcionalidades extra para generar código de MPI. Es por esto que por regla general Intel MPI es mejor que el resto, al poder tener acceso a las mejores optimizaciones para las CPUs de Intel. De habernos preocupado por performance Intel MPI habría sido nuestra primera elección. Consideramos que la mejor elección de MPI es una cuestión de ensayar varias implementaciones con HPL, y elegir la que experimentalmente arroje el mejor resultado.

3.3. Instalar BLAS

BLAS define un conjunto de rutinas para hacer operaciones con matrices y vectores. Es una interfaz, del mismo modo que lo es MPI, y tiene muchas implementaciones.

La implementación de BLAS que usamos es OpenBLAS [3], y la instalamos usando el manejador de paquetes.

Para el mejor performance habría que usar MKL [21]; es de las mejores opciones de BLAS (excepto en AMD), y en nuestra experiencia es la mejor opción con HPL. Más aún cuando se usa el compilador de Intel para compilar HPL.

3.4. Instalar y ejecutar HPL

Usamos la versión básica de HPL, que es la versión 2.3 portable que se encuentra en la página de Netlib [1].

La instalación de HPL se debe hacer manualmente. El paquete viene con instrucciones explícitas de lo que se debe hacer para compilar y ejecutar HPL. Básicamente se debe hacer lo siguiente:

1) Configurar el Makefile.

2) Compilar.
3) Configurar HPL.dat, que es el archivo de configuración donde está la parametrización de HPL.
4) Ejecutar HPL.

En la configuración del Makefile principal es en donde se relacionan las implementaciones de BLAS y MPI con la instalación de HPL; en este Makefile se le especifica al compilador dónde encontrar las librerías de MPI y BLAS que necesita HPL. El archivo Makefile principal que usamos debe estar adjunto con este documento y se llama `Make.Linux_MV0`.

Luego de la compilación, la única edición del archivo de configuración inicial de HPL fue hacer que corriera únicamente 1 test. No se optimizó absolutamente nada de parametrización. Adjuntamos el archivo de configuración que usamos, se llama `HPL.dat`.

La ejecución de HPL la hicimos usando `mpirun`, especificando que usara nuestros nodos y 2 procesos por nodo (`HPL.dat` determina la cantidad de procesos que se van a tener).

4. ¿Qué tan cerca del Heaven? Resultados de HPL

Se logró nuestro objetivo: tuvimos un resultado "pésimo". El resultado fue de $4.7153 \times 10^{-3} GFLOPs$. Con una eficiencia de 0.0000000708%, oficialmente estamos en el "Hell".

El archivo que respalda nuestro resultado está adjunto y se llama `HPL.out`.

5. Extra: ¿Cómo optimizar HPL?

La optimización de HPL se puede atacar desde varios frentes distintos. Algunos de ellos los exploramos a continuación.

5.1. Archivo de configuración

El grueso de la optimización de HPL se debe hacer desde el archivo de configuración. Los parámetros de más relevancia en el resultado son los siguientes:

- **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 [22], 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% [22].
- **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 [23].
- **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 hallar la mejor combinación de valores se pueden usar técnicas de auto-tuning que exploren muchas combinaciones de valores.

5.2. Compilación

A nivel de compilación también se pueden conseguir optimizaciones importantes. Algunas banderas de compilación que experiencialmente nos han probado eficacia son:

- **O3**: Le indica al compilador que busque optimizaciones de loops agresivamente, como habilitando Fusión, Block-Unroll-and-Jam, y colapsar condicionales.
- **xHost**: Le indica al compilador que use el set de instrucciones más alto que soporta la máquina.
- **fp-model fast=2**: Activa optimizaciones agresivas en instrucciones de punto flotante.

5.3. Comunicación

La comunicación entre nodos se puede mejorar al optimizar el uso de la red Infiniband. Hay librerías especializadas para el uso de redes ultra-rápidas. Un ejemplo de ellas es UCX [24], que es usada por la mayoría de las mejores implementaciones de MPI. Las implementaciones que hagan uso de las mejores características de la red tendrán un aumento de performance directo en las comunicaciones, y su configuración óptima casi nunca está disponible por defecto para el administrador del sistema.

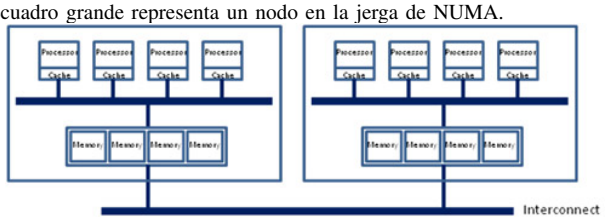
Hay tecnologías que operan sobre las redes que pueden mejorar sustancialmente el performance de la red. Una de ellas es Remote Direct Memory Access (RDMA). Sin RDMA, la información tiene que pasar por ambas CPUs de los nodos involucrados en comunicación para poder ser transferida exitosamente. RDMA es una tecnología que permite transferir información entre distintos nodos de una misma red disminuyendo a casi 0 el gasto de la CPU.

Este tipo de optimizaciones si bien son independientes de HPL pueden llegar a impactar drásticamente el performance de la aplicación. Es una muestra de que las optimizaciones de HPC no recaen únicamente en los retos algorítmicos y computacionales de cada aplicación, sino también en la configuración del hardware de cada supercomputador.

5.4. Afinidad - Optimizando para sistemas NUMA

En la mayoría de las máquinas modernas, el concepto de Non-Uniform Memory Access (NUMA) suele estar implicado en el diseño de arquitectura de cada máquina. Es normal encontrar hoy un servidor que tenga 2 CPUs, varios núcleos por cada CPU y varias unidades de memoria RAM (que se ven como una sola desde el SO). Este tipo de diseño

Figure 2. Ejemplo NUMA - Tomado de [25]. ”Processor” es núcleo. Cada recuadro grande representa un nodo en la jerga de NUMA.



implica una incongruencia en la velocidad del acceso a datos entre distintas unidades de procesamiento. Para entender mejor el concepto, miremos la figura 2.

En la figura vemos que cada núcleo tiene su memoria cache, y cada nodo NUMA tiene su grupo de memorias RAM. La velocidad de acceso a memoria no solamente está ligado a la jerarquía de memoria, sino también al lugar donde se encuentre el dato que se quiera acceder. El acceso a memoria no es uniforme, puesto que para un núcleo del socket de la izquierda, acceder un dato que yace en la memoria RAM del socket de la derecha toma mucho más tiempo que acceder uno que yazca en su mismo socket.

Una de las técnicas más comunes para optimizar aplicaciones de HPC es usar ”binding” o ”pinning”. Se puede optimizar el performance de una aplicación multi-proceso usando técnicas de ”binding” o ”pinning” para hacer que un proceso solamente se pueda ejecutar en un solo núcleo físico. Por ejemplo, hacer que el proceso #0 solamente se pueda ejecutar en el núcleo #7, logrando que hayan menos cambios de contexto.

El uso de ”binding” es ampliamente soportado por las implementaciones de MPI, y todas las que conocemos tienen opciones para configurar ”binding” [26] [27] [28] [29]. Es tan común que hasta Slurm tiene amplio soporte para CPU binding [30].

6. Lecturas recomendadas

El capítulo 11 de The Student Supercomputer Challenge Guide [31] es un buen texto introductorio para optimizar HPL.

El capítulo 4 de High Performance Computing de Sterling et. al. [32], tiene información sobre el benchmarking en el mundo del HPC. También tiene una sección donde trata HPL brevemente.

Intel tiene un centro de recursos web muy amplio que cubre muchas técnicas de optimización de software. Tiene también una guía para mejorar el performance del cluster [33]. Por otro lado también tiene su propia versión optimizada de HPL [34]. La guía de optimización aplicaciones en arquitecturas NUMA es muy relevante a pesar de ser del 2011 [25]. También tiene guías de optimización para Intel MPI [35].

Mellanox es una compañía de interfaces Infiniband que hace unos años fue adquirida por Nvidia. Tienen lecturas que profundizan en la optimización de redes Infiniband. Estas lecturas pueden ser de ayuda en la etapa de configuración de cluster [36].

References

[1] J. D. A. C. P. L. Antoine Petitet, Clint Whaley, “Hpl 2.3 - a portable implementation of the high-performance linpack.” [Online]. Available: https://www.netlib.org/benchmark/hpl/

[2] “CENTOS,” 2021. [Online]. Available: https://www.centos.org/

[3] Zhang Xianyi, Martin Kroeker, “OpenBLAS,” 2021. [Online]. Available: http://www.openblas.net/

[4] “MPICH,” 2021. [Online]. Available: https://www.mpich.org/

[5] GNUTools, “Gcc, the gnu compiler collection.” [Online]. Available: https://gcc.gnu.org/

[6] Red Hat Customer Portal, “RHEL 8 Managing File Systems, CHAPTER 4. EXPORTING NFS SHARES,” 2021. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/managing_file_systems/exporting-nfs-shares_managing-file-systems#configuring-an-nfsv4-only-server_exporting-nfs-shares

[7] © Intel Corporation, “Intel Developer Guide and Reference, Code Generation Options, xHost,” 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/documentation/cpp-compiler-developer-guide-and-reference/top/compiler-reference/compiler-options/compiler-option-details/code-generation-options/xhost-qxhost.html

[8] Red Hat, “Reasons to Migrate from NFS v3 to v4/ v4.1,” 2018. [Online]. Available: https://archive.fosdem.org/2018/schedule/event/nfs3_to_nfs4/attachments/slides/2702/export/events/attachments/nfs3_to_nfs4/slides/2702/FOSDEM_Presentation_Final_pdf.pdf

[9] © Intel Corporation, “Tecnología de las extensiones del conjunto de instrucciones Intel®,” 2020. [Online]. Available: https://www.intel.la/content/www/xl/es/support/articles/000005779/processors.html

[10] Hans Meuer, Erich Strohmaier, Jack Dongarra, Horst Simon, Martin Meuer, “Top 500 JUNE 2021,” 07 2021. [Online]. Available: https://top500.org/lists/top500/2021/06/

[11] ——, “Top 500 Statistics EFFICIENCY,” 07 2021. [Online]. Available: https://top500.org/statistics/efficiency-power-cores/

[12] Wiki Chip, “Floating-Point Operations Per Second (FLOPS),” 2021. [Online]. Available: https://en.wikichip.org/wiki/flops

[13] Message Passing Interface Forum, “MPI: A Message-Passing Interface Standard Version 4.0,” 2021. [Online]. Available: https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf

[14] Netlib, “BLAS (Basic Linear Algebra Subprograms),” 2021. [Online]. Available: https://www.netlib.org/blas/

[15] SlothParadise, “How to use MPI without NFS,” 2016. [Online]. Available: https://www.slothparadise.com/use-mpi-without-nfs/

[16] Red Hat, Inc, “GLUSTER,” 2021. [Online]. Available: https://www.gluster.org/

[17] OpenSFS and EOFS, “Lustre,” 2021. [Online]. Available: https://www.lustre.org/

[18] “Open MPI: Open Source High Performance Computing,” 2021. [Online]. Available: https://www.open-mpi.org/

[19] “MVAPICH2: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE,” 2021. [Online]. Available: https://mvapich.cse.ohio-state.edu/

[20] “Intel® MPI Library,” 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/mpi-library.html#gs.7ipg9i

[21] I. Corporation, “Intel® oneAPI Math Kernel Library,” 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html#gs.7inc96

[22] A. Community, *The Student Supercomputer Challenge Guide*. Springer, 2018.

[23] Netlib, "Frequently Asked Questions on the Linpack Benchmark and Top500," 2007. [Online]. Available: https://www.netlib.org/utk/people/JackDongarra/faq-linpack.html#_For_HPL_what_block%20size%20NB%20should%20I

[24] UCF Consortium, "Unified Communication X," 2021. [Online]. Available: https://www.openucx.org/

[25] Intel Corporation, "Optimizing Applications for NUMA," 2011. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/optimizing-applications-for-numa.html

[26] Software in the Public Interest , "FAQ: General run-time tuning," 2019. [Online]. Available: https://www.open-mpi.org/faq/?category=tuning#paffinity-defs

[27] Intel Corporation , "Process Pinning," 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/documentation/mpi-developer-reference-linux/top/environment-variable-reference/process-pinning.html

[28] "Using the Hydra Process Manager," 2019. [Online]. Available: https://www.open-mpi.org/faq/?category=tuning#paffinity-defs

[29] THE CENTER FOR HIGH PERFORMANCE COMPUTING, "Message Passing Interface." [Online]. Available: https://www.chpc.utah.edu/documentation/software/mpilibraries.php

[30] SchedMD, "Resource Binding," 2020. [Online]. Available: https://slurm.schedmd.com/resource_binding.html

[31] A. Community, *The Student Supercomputer Challenge Guide*. Science Press Beijing, 2018.

[32] T. Sterling, M. Anderson, and M. Brodowicz, *High performance computing*. Cambridge Morgan Kaufmann 2018, 2018.

[33] Intel Corporation, "Improving Performance of Your Cluster," 6 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/documentation/onemkl-linux-developer-guide/top/intel-oneapi-math-kernel-library-benchmarks/intel-distribution-for-linpack-benchmark/improving-performance-of-your-cluster.html

[34] ——, "Overview of the Intel® Distribution for LINPACK* Benchmark," 6 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/documentation/onemkl-linux-developer-guide/top/intel-oneapi-math-kernel-library-benchmarks/intel-distribution-for-linpack-benchmark/overview-of-the-intel-distribution-for-linpack-benchmark.html

[35] I. Corporation, "Tuning the Intel® MPI Library: Basic Techniques," 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/tuning-the-intel-mpi-library-basic-techniques.html

[36] M. Technologies, "Performance Tuning for Mellanox Adapters," 2019. [Online]. Available: https://community.mellanox.com/s/article/performance-tuning-for-mellanox-adapters