Reporte Técnico:Process Assignment in Multi-core Clusters Using Job Assignment Algorithm

Universidad Nacional San Agustin
Escuela Profesional de Ciencia de la Computación
Estudiante: Katerine Cruz Valdivia
Erika Tamo Turpo
Ingrid Espinel Quispe
kcruzv@unsa.edu.pe, iespinel@unsa.edu.pe

***Resumen*—Los modernos sistemas de clústeres de alto rendimiento para procesamiento en paralelo emplean procesadores de múltiples núcleos y redes de interconexión de alta velocidad. El mapeo eficiente de los procesos de una aplicación paralela en los núcleos de tal sistema de clúster, desempeña un papel vital en la mejora del rendimiento de esa aplicación. La aplicación paralela se puede modelar como un gráfico ponderado que muestra la comunicación entre los procesos de esa aplicación. Dicho gráfico se puede construir con la ayuda de herramientas de perfilado. El hardware del clúster también se puede modelar como un gráfico, mediante la recopilación de detalles del hardware utilizando la herramienta HWLOC(cual provee información sobre los procesadores de los nodos). El enfoque basado en el peso máximo se puede utilizar para incrustar el gráfico de la aplicación en el gráfico de hardware de clúster. El enfoque propuesto se implementa bajo un sistema de clúster y se prueba utilizando una aplicación paralela MPI de referencia. El rendimiento de la aplicación paralela, que se mapea utilizando el enfoque propuesto es mejor que el emparejamiento utilizando los enfoques heredados empaquetados y round robin de la biblioteca MPI**

I. Introducción al Problema

A pesar de que la velocidad de los clústeres de alto rendimiento está aumentando, la eficiencia y el tiempo de ejecución de las aplicaciones paralelas deben mejorarse siempre. Como los clústeres de múltiples núcleos se utilizan generalmente para ejecutar estas aplicaciones paralelas, una ubicación adecuada e inteligente de los procesos basada en su comportamiento de comunicación puede mejorar significativamente el rendimiento de la aplicación. Además, las propiedades del hardware del clúster subyacente, como el ancho de banda entre los procesadores, la ubicación de la caché, etc., deben considerarse para mejorar el rendimiento de las llamadas de paso de mensajes.

II. Descripción del Problema

Este documento apunta a la ubicación del proceso para abordar el problema de la localidad que proviene de la forma en que se intercambian los datos entre los procesos de una aplicación paralela, ya sea a través de la red o a través de la memoria. El método de asignación de trabajos se utiliza para encontrar una asignación adecuada para los procesos MPI en función de su patrón de comunicación e información de hardware para disminuir la comunicación entre nodos y, por lo tanto, aumentar la eficiencia y el rendimiento. Los procesos con un gran costo de comunicación entre procesos generalmente se asignan al mismo nodo para compartir la misma localidad o los nodos que están cerca entre sí para reducir el costo de la comunicación.

III. Estado del Arte

El problema de asignación de trabajo es uno de los problemas fundamentales de optimización en matemáticas. Consiste en encontrar una coincidencia perfecta de peso mínimo (o una combinación de peso máximo) en un grafo bipartito ponderado.En otras investigaciones lo resolvieron como lo siguiente:

- **Modificando MPI :**En [4], el autor desarrolló una biblioteca modificada de primitivas de comunicación colectiva MPI (barrera, transmisión, reducción y recopilación), que están optimizadas para sistemas de área amplia al minimizar la cantidad de datos comunicados a través de los enlaces. El trabajo en [5] optimiza las operaciones de comunicación punto a punto, considerando la información de la topología del hardware, al clasificar virtualmente los procesos MPI, de modo que más pares de procesos comunicantes se mantienen físicamente cerca uno del otro.
- **Usando perfilador :** El enfoque guiado por perfil para obtener una locación adecuada de los procesos.En los siguientes trabajos los autores intentan asignar los procesos paralelos a los procesadores adecuados para minimizar el costo total de la comunicación entre los procesadores del clúster utilizando el **concepto de localidad y ubicando los procesos de comunicación extensiva en el mismo localidad**.
  - MPIPP [5]encuentra la máquina en la que se puede colocar un proceso, no el procesador, por lo que no optimiza completamente el proceso de colocación.TREEMATCH [6] utiliza algoritmos para encontrar el conjunto independiente de ponderaciones mínimas del grupo de procesos. Como el procedimiento es NP-Duro, se utilizan pocas heurísticas para la optimización.

- **Mapeando con grafos :** El problema de mapeo se puede modelar como un problema de integración de grafos que puede resolverse fácilmente mediante la partición de grafos. Estas técnicas son utilizadas por varios autores en Chaco [7], METIS [8] y SCOTCH [9].
- **Algoritmo Hungaro :** El algoritmo húngaro es uno de los algoritmos simplex primarios dados por Harold Kuhn en [10], [11] que se han utilizado para resolver el problema de asignamiento.

IV. Solución del problema

En la literatura se propusieron varios métodos para mejorar el rendimiento de las aplicaciones paralelas en grupos de múltiples núcleos. Un enfoque es optimizar el rendimiento de la comunicación MPI.

Para resolver el problema del mapeo, algunos sistemas como Chaco [5], METIS [6] y SCOTCH [7] utilizan técnicas de partición de grafos, y Harold Kuhn [8, 9] utiliza el método húngaro.

IV-A. Descripción de la solución

Se propone un metodo propuesto ejecutados en tres pasos como se muestra en la VIII :

- Un perfilador de la aplicación paralela: Recopila información sobre una aplicación paralela, nos proporciona la cantidad y el tamaño de los mensajes enviados entre dos procesos.
- Recopilación de información de hardware: HWLOC es una herramienta que proporciona una abstracción portátil de la topología jerárquica de las arquitecturas del computador como los nodos de memoria NUMA, sockets, cachés compartidos, núcleos y unidades de procesamiento.
- Aplica el algoritmo de asignación de trabajo para obtener el mapeo adecuado: Los procesos paralelos basados en MPI se pueden mapear de dos maneras:
  - Enlace de Recursos: el usuario debe seleccionar un procesador para ejecutar un proceso bajo consideración para disminuir el costo general de comunicación.
  - Métodos de reordenamiento de rango: los rangos de los procesos se reasignan en función de la información obtenida del primer y el segundo paso.

IV-B. Problemas encontrados en la solución

En el paper propuesto no se encontraron problemas en la solución pero al momento de replicar tuvimos problemas en instalar un perfilador para MPI por las versiones MPIP 3.4.1 y cambiando a la versión de mpich 2.2 se podria trabajar con está libreria pero como *integer sorting* trabaja con mpich 3 no obtamos por está solución.
Nuevamente al revisar la documentación por fin resolvimos de la libreria de ubuntu y las versiones de mpich 3.2 y MPIP 3.4
.

Obtener información de hwlock de cada computadora es muy complicado hacerlo al obtenerlo desde el nodo maestro.

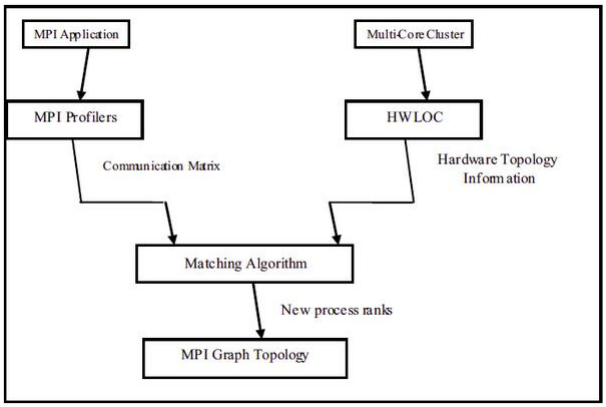


Figura 1. Arquitectura de la Solución

IV-C. Caracteristicas de la infraestructura

HWLOC : Su objetivo es ayudar a las aplicaciones proporcionando las llamadas API necesarias para recopilar información sobre el hardware informático subyacente(memoria,sockets,caches compartidos).Se obtiene la arquitectura de las computadores estructurado en forma de árbol y se guarda en archivo. En este archivo la primera línea contiene n, que indica el número de niveles en la arquitectura del hardware, y la segunda línea contiene n valores enteros que indican la aridad (número de hijos) de cada nivel a partir de la raíz.

3
2 4 2

En la anterior imagen contiene el hadware 3 nieveles,el primer level tiene aridad 2 que significa que la raiz tiene dos hijos.En el siguiente, el level 2 tiene por cada nodo en este nivel 4 hijos y asi suecesivamente al 2.

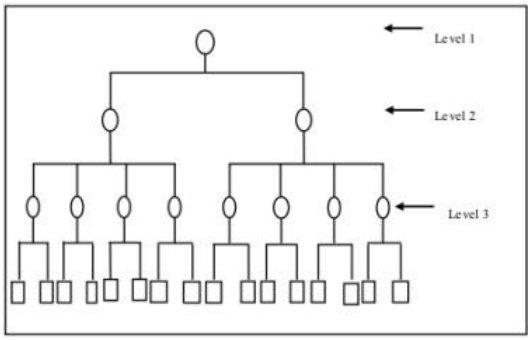


Figura 2. Topologia de Hwlock grafica

Matriz de Costos : Está matriz se va a sacar desde el perfiladores de MPI usualmente es la cantidad de mensajes.

Una desventaja de lo propuesto es que el algoritmo principal se demora $O(n^4)$ donde 'n' es la cantidad de procesadores.

IV-D. Implementación

Este paper se implemento en:

- c++.
- MPI(3 computadoras master y esclavo)
- Hwloc
- Mpi Profiler
- Mapeo (Metodo Hungaro
- Representacion(MPI Dist graph create)

Los procesos con más costos de comunicación entre procesos pueden asignarse al mismo núcleo. Si no es posible, dichos procesos se asignan a las unidades de procesamiento más cercanas.Entonces estamos con un problema de asignamiento, tenemos n tareas que queremos asignar a n hilos, tal que su costo total sea minima.[2]

El método **húngaro** es un algoritmo que encuentra una asignación óptima para una matriz de costo dada. Su algoritmo es el siguiente[2]:

1. Reste la entrada más pequeña en cada fila de todas las entradas de su fila.
2. Reste la entrada más pequeña en cada columna de todas las entradas de su columna.
3. Dibuje líneas a través de filas y columnas apropiadas para que se cubran todas las entradas de cero de la matriz de costos y se use el número mínimo de dichas líneas.
4. Prueba de Optimalidad:
   - Si el número mínimo de líneas de cobertura es igual al número de procesos, es posible una asignación óptima de cero, luego se completa el algoritmo.
   - Si el número mínimo de líneas de cobertura es menor que el número de procesos, aún no es posible una asignación óptima de ceros. En ese caso, proceda al Paso 5.
5. Determine la entrada más pequeña no cubierta por ninguna línea. Reste esta entrada de cada fila descubierta y luego agréguela a cada columna cubierta. Vuelva al paso 3.

El pseudocodigo siguiente(IV-D) muestra el algoritmo utilizado para calcular la nueva disposición para los procesos. En este sentido, los procesos se combinan de forma iterativa en función de su costo de comunicación utilizando técnicas de comparación de grafos. Con el fin de minimizar el costo total de comunicación, los procesos que tienen un mayor costo de comunicación entre procesos se emparejan primero. Después de encontrar una coincidencia óptima, se calcula la nueva matriz de comunicación agregada. Esta nueva matriz de comunicación se utiliza en la siguiente iteración.

Mapping Algorithm

```
Input : CM - Cost matrix
N      Size of cost matrix
arity      array storing arities of
different levels
Output : new_result - assignment result

old_result = NULL
for i -> num_of_level to 0 down by 1
   for j -> 1 to arity_at_level_i up by j
       *2
       result = assignment(CM, N, arity)
       aggregated_mat = aggregatematrix(CM
           ,
       result, N)
       if old_result is not NULL
       new_result = aggregateresult(
           old_result,
       new_result, N)
       oldresult = new_result
      CM = aggregated_mat
   end loop j
end loop i
```

	0	1
0	0	272
1	272	0

Cuadro I
TABLA 3 : MATRIZ DE COSTOS DESPUÉS DE LA ITERACIÓN 2

TABLE I SAMPLE COSTMATRIX

	0	1	2	3	4	5	6	7
0	0	100	19	12	19	11	2511	223
1	100	0	12	12	25	28	223	1412
2	19	12	0	223	251	1584	17	10
3	12	12	223	0	1995	112	17	17
4	19	25	251	1995	0	158	17	12
5	11	28	1584	112	158	0	25	19
6	2511	223	17	17	17	25	0	100
7	223	1412	10	17	12	19	100	0

Figura 3. Tabla 1 : Ejemplo de matriz de costos

Por ejemplo, el costo de la matriz de costo en Tabla I y con una red de interconexión de hardware de estructura de árbol binario de 3 niveles, el resultado de la primera iteración del algoritmo de mapeo m es (0, 6), (1, 7), (2, 5) y (3, 4). Por lo tanto, después de llamar al algoritmo de asignación m, la nueva matriz de costos agregada resultante se muestra en la Tabla II. La Fig.IV-D muestra una matriz de cálculo de la matriz de costos agregados.

Aggregate Cost Matrix

```
Input: old_matrix, old_result, n, arity
Output: new aggregated_matrix
  allocate and initialize new_matrix
  for i->0 to n/arity increment by 1
    for j->0 to n/arity increment by 1
      new_matrix[i][j] = 0
      if i is not equal to j
        for k->0 to arity increment by 1
          for p->0 to arity increment by
              1
            l = old_result[i][k]
            m = old_result[j][p]
            new_matrix[i][j] +=
                old_matrix[l][m]
        end for
      end for
    end for
  end for
```

Los nuevos grupos de procesos de la tercera iteración son (0, 6, 1, 7) y (2, 5, 3, 4). La matriz de costos correspondiente

TABLE II. COST MATRIX AFTER ITERATION 1

	0	1	2	3
0	0	646	72	65
1	646	0	69	66
2	72	69	0	744
3	65	66	744	0

Figura 4. Tabla 2 : MATRIZ DE COSTOS DESPUÉS DE LA ITERACIÓN 1

se muestra en la Tabla III. Por lo tanto, la nueva disposición para colocar los procesos es (0, 6, 1, 7, 2, 5, 3, 4) en lugar de (0, 1, 2, 3, 4, 5, 6, 7). La salida del algoritmo de mapeo se usa entonces como una entrada al algoritmo m para la reasignación de rangos de proceso que se muestra en la Fig. IV-D.

Re-Assignment of Process Ranks

```
Input: file – adjacency_matrix
       file      cost_matrix
Output: new communicator for MPI
    processes
    Initialize MPI
    if my_rank is 0
        initialize array pointers:
            sources, degrees,
            weights and destinations as
                NULL
        enter_graph_values ( num_prcs,
            destinations, sources,
                degrees,
            adjacency_matrix)
        MPI_Dist_graph_create(
            MPI_COMM_WORLD,
            comm_size, sources, degrees,
            destinations, MPI_UNWEIGHTED,
            MPI_INFO_NULL, reorder,
            &comm_topology)
    else
        MPI_Dist_graph_create(
            MPI_COMM_WORLD,
          0, NULL, NULL, NULL,
              MPI_UNWEIGHTED,
          MPI_INFO_NULL, reorder,
          &comm_topology)
    if comm_topology != MPI_COMM_NULL
        comm_to_use = comm_topology
    else
        comm_to_use = MPI_COMM_WORLD
    MPI_Comm_rank ( comm_to_use, &my_rank
        )
    Rest of the MPI Code
```

Al utilizar esta información de disposición modificada, al reasignar los rangos de proceso, se crea un nuevo comunicador con una nueva topología de comunicación. Por lo tanto, los rangos del proceso se reordenan utilizando la llamada de biblioteca MPI_Dist_graph_create (...). Esta llamada se realiza justo después del paso de inicialización y antes de que los datos de la aplicación se carguen en los procesos de MPI. Las operaciones de comunicación subsiguientes se realizan utilizando este nuevo comunicador que ofrece un mejor rendimiento.

El administrador de procesos de MPI generalmente programa los procesos de forma rotatoria o empaquetado. Pero debido al reordenamiento de rango, el administrador del proceso tiene que usar los nuevos rangos asignados bajo el nuevo comunicador.

Todos estos pseudocodigos ya implementados se pueden encontrar en https://github.com/StarUnu/ProyectoAP-Asignamiento-Tareas-en-un-Cluster

IV-E. Experimentación

Para el primer paso es instalar whloc en las tres maquinas 5, 6, 7

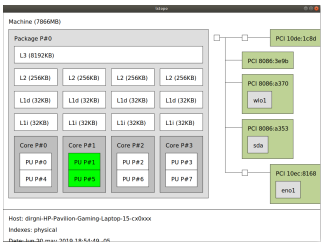


Figura 5. Topologia de Hwlock Maquina Nro 1

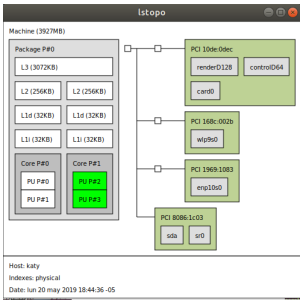


Figura 6. Topologia de Hwlock Maquina Nro 2

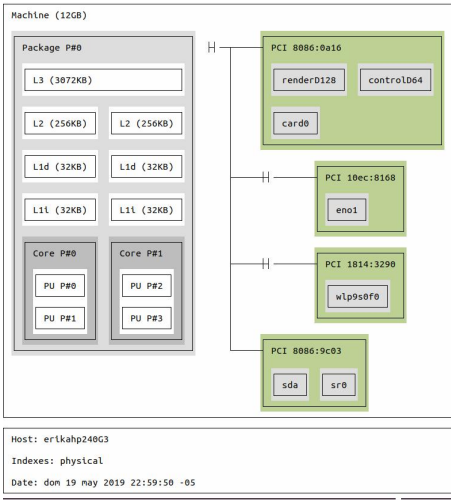


Figura 7. Topologia de Hwlock Maquina Nro 3

V. Análisis de la solución

En los resultados se muestran que el método propuesto disminuye considerablemente el tiempo de ejecución de la aplicación en comparación con las políticas heredadas de MPI y RR.

Se muestra nuevo método que calcula la ubicación de un proceso de la aplicación adaptada para una computadora. Se basa tanto en el patrón de comunicación de la aplicación como en la arquitectura de la máquina.

El objetivo del algoritmo propuesto es minimizar el costo general para que la aplicación se ejecute en paralelo. Por lo tanto, los procesos con más costos de comunicación entre procesos pueden asignarse al mismo núcleo. Si no es posible, dichos procesos se asignan a las unidades de procesamiento más cercanas.Para esto se usa el algoritmo *Hungarean Assignment Algorithm* que su objetivo es tener la menor matriz de costo.Para despues pasar el *Aggreate Cost Matrix* es minimizar el costo general para que la aplicación se ejecute en paralelo. Por lo tanto, los procesos con más costos de comunicación entre procesos pueden asignarse al mismo núcleo.

Si no es posible, dichos procesos se asignan a las unidades de procesamiento más cercanas.Con el fin de minimizar el costo total de comunicación, los procesos que tienen un mayor costo de comunicación entre procesos se emparejan primero. Después de encontrar una coincidencia óptima, se calcula la nueva matriz de comunicación agregada. Esta nueva matriz de comunicación se utiliza en la siguiente iteración.Después se usa estos datos mientras se reasignan los rangos del proceso, se crea un nuevo comunicador con una nueva topología de comunicación. Por lo tanto, los rangos del proceso se reordenan utilizando la llamada de biblioteca MPI_Dist_graph_create (...)

VI. Comparación de los resultados con el Articulo GUIA

RESULTADOS DE LA GUIA

La plataforma utilizada para la experimentación es un clúster de 1 + 8 nodos que ejecuta el clúster Rocks 6.1.1 sobre CentOS 6.5. La parte frontal del clúster es HP Proliant DL380p Gen8, 2 x procesador Intel Xeon E5-2640 con reloj de 2,5 GHz, 6 núcleos, 15 MB de caché y 64 GB de RAM. Cada nodo contiene 1 x procesador Intel Xeon E5-2640 con reloj de 2.5 GHz, 6 núcleos, 15 MB de caché, 16 GB de RAM y 2 * 300 GB de disco duro. Para ejecutar los algoritmos implementados se usa el siguiente software: compilador GNU C, Open MPI, HWLOC y MPI en perfiladores incorporados.

La aplicación seleccionada para comparación es IS (Integer Sort) de la suite de referencia NPB de la NASA. Esta aplicación está más orientada a la comunicación y tiene algún patrón regular para la comunicación. El tiempo de ejecución se considera como un parámetro de comparación para las versiones de la aplicación con y sin reordenación de los procesos. Los resultados comparativos se muestran en la Tabla IV. y la gráfica de líneas correspondiente se muestra en la Fig.9.

De la tabla IV, está claro que inicialmente el algoritmo propuesto está tomando más tiempo para un número menor de procesos (8 o 16) debido a la sobrecarga de calcular la disposición adecuada y la creación de topología. Pero a medida que aumenta el número de procesos (más de 16), la fracción del tiempo de sobrecarga consumido por el algoritmo propuesto disminuye gradualmente. El porcentaje de mejora en el tiempo de ejecución se muestra en la última columna de la Tabla IV.

Number of Process	Time without reordering (seconds)	Time with Reordering (seconds)	% improvement
8	0.0225	0.05614	-149%
16	17.543	18.01022	-2.66%
32	98.201	74.32241	+24.31%
64	262.736	189.23373	+27.97%
128	617.531	425.23948	+31.13%

Figura 8. TABLA IV. TIEMPO DE EJECUCIÓN CON Y SIN RANGO REORDENADO DE PROCESOS

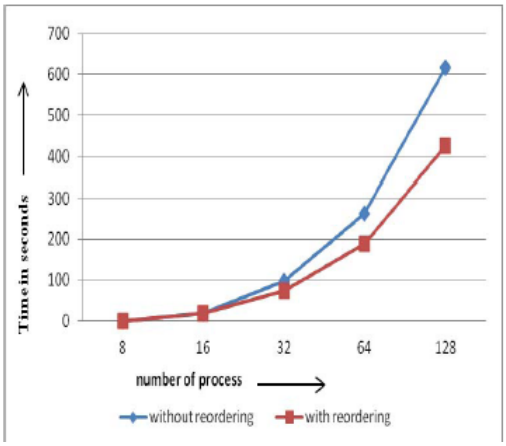


Figura 9. Comparación del tiempo de ejecución con y sin reordenamiento de procesos

Utilizando la misma libreria que ejecuta el articulo de referencia en una maquina obtenemos lo siguiente con 4 procesadores.

```
erika@erikahp240c3:~/Música/pobcpp-master/is-nas/IS$ mpiexec -n 4 ./a.out
 numero de procesos  4
 numero de procesos  4


 NAS Parallel Benchmarks 2.4 -- IS Benchmark

 Size:  65536  (class C)
 Iterations:   10
 Number of processes:     4
 numero de procesos  4
 numero de procesos  4
time 0.006770
time 0.006284
time 0.007511
time 0.007493
time 0.000693
time 0.000782
time 0.000793
```

Figura 10. Integer Sorting Salida

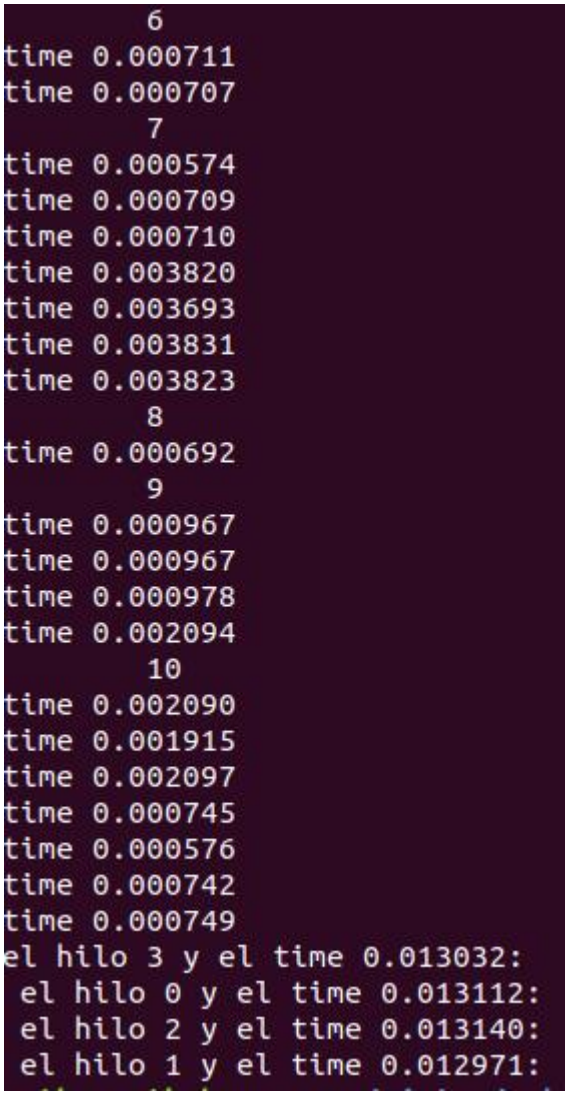


Figura 11. Integer Sorting Salida

Logicamente nuestros resultados es mucho mayor el tiempo con 4 procesadores ya que ellos precisan que usan supercomputadores mientras que nosotras usamos computadoras e conexion mediante wifi, que es mucho más lenta que por cable.

Apesar de estar en examenes logramos encontrar el perfilador y el algoritmo que ejecuta el articulo aun tenemos problemas en ejecutar el paper por incompatibilidad de computadoras.Tambien logramos sacar las arquitecturas de las computadoras y los nodos.

Todo lo que hemos hecho se encuentra en el siguiente link: https://github.com/StarUnu/ProyectoAP-Asignamiento-Tareas-en-un-Cluster

VII. Conclusiones

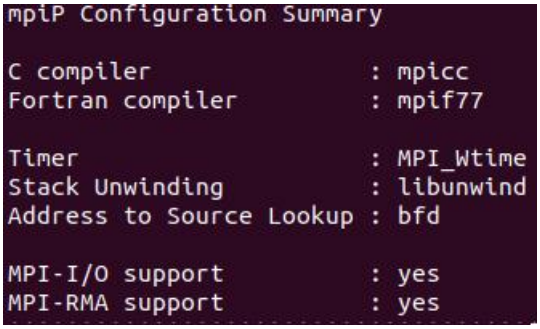
- El acceso a los datos y la comunicación es un problema clave para obtener un buen rendimiento del hardware subyacente.
- La velocidad de comunicación entre las unidades informáticas no solo depende de sus ubicaciones (debido al tamaño de la memoria caché, la jerarquía de la memoria, la latencia y el ancho de banda de la red, la topología, etc.), observar los patrón de comunicación entre los procesos.
- Para poder usar librerias se tiene que fijar de las dependencias que tiene como compiladores,libererias entre otras cosas.

VIII. Propuesta y/o aporte al tema trabajado.

En el analisis qu se realizo en el paper se vio que existen diferentes tipos de métricas, se pueden utilizarse y para optimizar, incluir la potencia de cálculo del nodo y su ancho de banda como un factor decisivo en la matriz de costos.

La solución propuesta para la comunicación en un cluster se basa principalmente en **MPI,perfilador de mpiP,libreria hwloc y el algoritmo propuesto basado en Hungaro**.

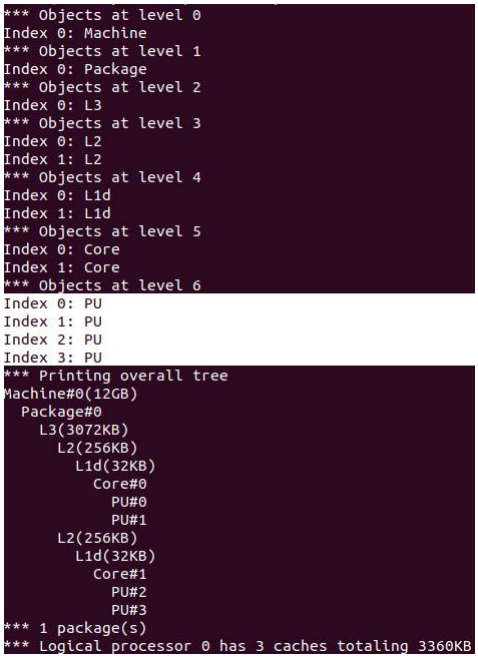
- **MPI:** Se utiliza mpich3.3, este a la vez usa el compilador **GCC** y **G++**.
- **Perfilador de mpiP :** Principalmente se basa en lo siguiente:



BFD(Brute Force Detection) : Es un script que se ejecuta en su servidor Linux y comprueba los archivos de registro en busca de errores de autenticación.
Libunwind : Tiene las siguientes caracteristicas:

- Hace que sea trivial en la implementación aspectos de manipulación de la pila del manejo de excepciones.
- Hace que sea trivial para los depuradores generar la cadena de llamadas (backtrace) de los hilos en un programa en ejecución.
- Muestra en un **hilo de ejecución su cadena de llamadas**. Por ejemplo, es útil para mostrar mensajes de error (para mostrar cómo se produjo el error) y para el monitoreo / análisis del rendimiento.

HWLOCK Que de cada nodo cliente con la ayuda de un programa hwlock_prueba1.c lo ejecutamos a cada cliente para sacar su *.txt y ponerlo como entrada hacia nuestra matriz de costos.



Para nuestra siguiente articulo se mejorara

- Mejora el acceso a los datos y la comunicación(por ejemplo cluster mediante conexion cableada aumenataria la velocidad), es un problema clave para obtener un buen rendimiento del hardware subyacente.
- Con el método nuevo se calcula la ubicación de un proceso de la aplicación adaptada para una computadora. Se realiza con un patrón de comunicación de la aplicación como en la arquitectura de la máquina.

REFERENCIAS

[1] Jorge Gabriel Hoyos-Pineda. Álex Puertas-González. .La programación funcional y las arquitecturas multicore: estado del arte.Universidad Santo Tomás.2015

[2] Derek Bruff, ”The Assignment Problem and the Hungarian Method”,Math 20 –Harvard Mathematics Department,2005.

[3] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, Raoul A. F. Bhoedjang, MAGPIE: MPI’s Collective Communication,Atlanta,1999

[4] J. L. T raff,Implementing the MPI process topology mechanism, Baltimore,2002

[5] H. Chen, W. Chen, J. Huang, B. Robert, and H. Kuhn, ”MPIPP: An Automatic Profile-Guided Parallel Process Placement Toolset for SMP Clusters and Multiclusters”,2006

[6] Emmanuel Jeannot, Guillaume Mercier, Francois Tessier, ”Process Placement in Multicore Clusters: Algorithmic Issues and Practical Techniques”, IEEE Transaction on Parallel and Distributed Systems, vol. 25, no. 4, pp. 993-1002, 2014

[7] B. Hendrickson and R. Leland, ”The Chaco User’s Guide: Version 2.0”,Sandia Nat’l Laboratory, 1994.

[8] G. Karypis, V Kumar, ”METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0”, technical report 1995

[9] F. Pellegrini, ”Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs”,Mayo 1994

[10] Harold W. Kuhn, ”The Hungarian Method for the assignment problem”, Naval Research Logistics Quarterly, pp. 83–97, 1955.

[11] Harold W. Kuhn, ”Variants of the Hungarian method for assignment problems”, Naval Research Logistics Quarterly, pp. 253–258, 1956.