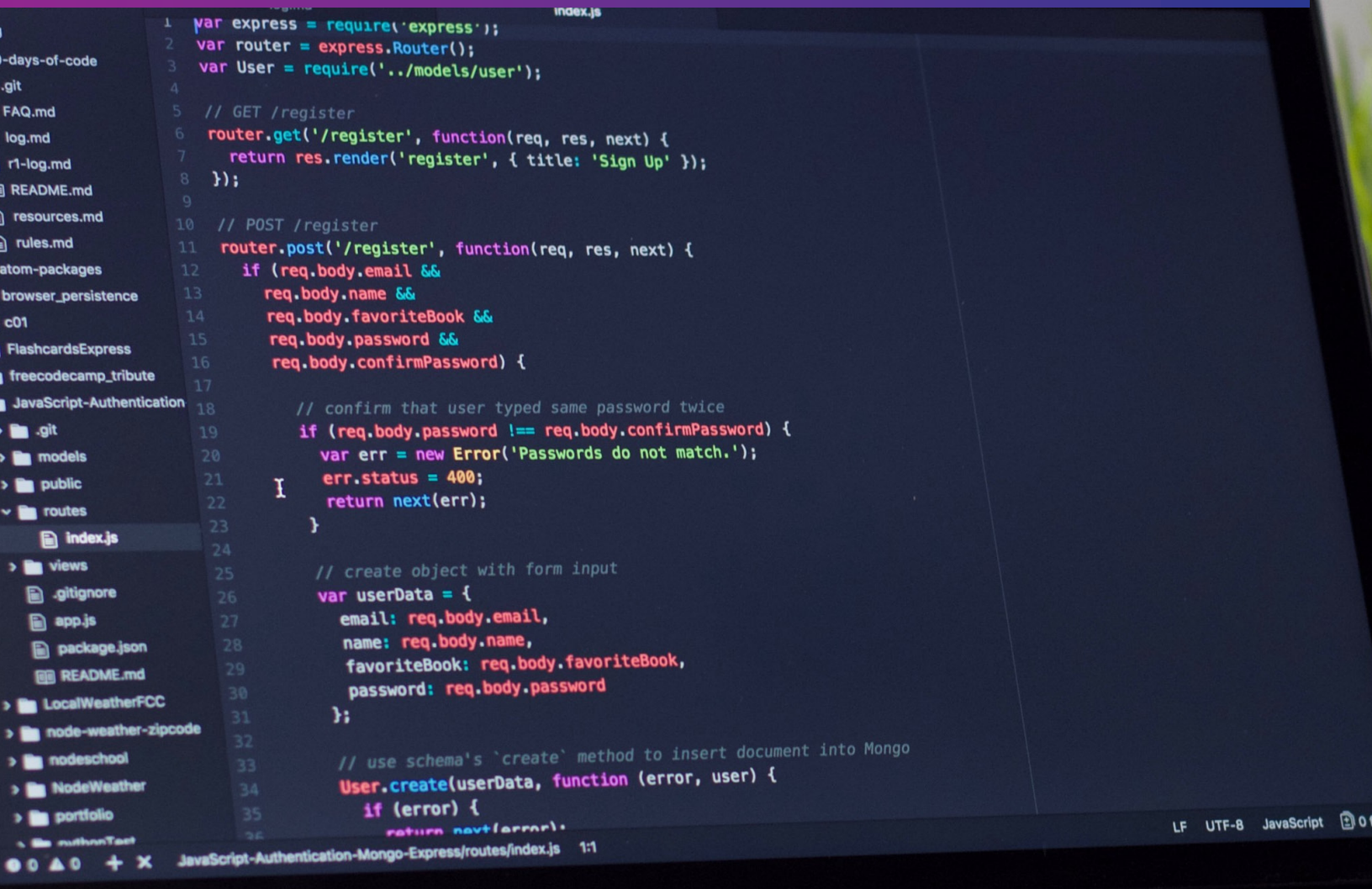
Uso de Programación paralela en clúster de alto rendimiento



Ing. Gaddiel Fredy Flores Arteaga[1], Instituto Tecnológico de Acapulco, Acapulco Gro., México. CP 39902 gadflores@gmail.com, Dr. Eduardo De la Cruz Gámez[2], Instituto Tecnológico de Acapulco, Acapulco Gro., México. CP 39902, gamezeduardo@yahoo.com, M.T.I. Eloy Cadena Mendoza[3], Instituto Tecnológico de Acapulco, Acapulco Gro., México. CP 39902 eloy_cadena@yahoo.com, M.C. Francisco Javier Gutiérrez Mata[4] Instituto Tecnológico de Acapulco, Acapulco Gro., México. CP 39902, fcomata84@hotmail.com.

gadflores@gmail.com
(222) 2996513

Resumen.
El presente artículo demuestra una comparación del tiempo de ejecución de un programa para buscar todos los números primos existentes dentro de un intervalo dado mediante la implementación de librerías MPI y Open MPI, en una agrupación de computadoras (clúster) de alto rendimiento, el cuál, se compara contra el rendimiento de una PC para poder demostrar la eficiencia de los tiempos de respuesta en ejecución de los procesos llevados a cabo en uno y otro sistema. Dentro del presente trabajo se menciona como marco teórico el clúster, una arquitectura básica de componentes, además de información relevante a ser implementada en trabajo a futuro con algoritmos en paralelo.

Palabras clave: Clúster; Programación Paralela; Procesamiento de información.

Abstract.
This paper shows a comparison of the execution time of a program to search all the existing prime numbers within a given interval through the implementation of MPI and Open MPI libraries, in a high performance cluster, which it is compared against the performance of a PC to be able to demonstrate the efficiency of the response times in execution of the processes carried out in both systems. Within this work is mentioned as a theoretical framework the cluster, a basic architecture of components, as well as relevant information to be implemented in future work with parallel algorithms.

Keywords: Cluster; Parallel Programming; information processing.

I. INTRODUCCIÓN

Desde los años 50, con la aparición de los primeros equipos computacionales se comenzó a utilizar este término “procesamiento en paralelo”, en el año 1956 IBM es el primero que le apuesta a lanzar un proyecto llamado 7030 para construir una supercomputadora para la empresa Los Alamos National Laboratory, teniendo como objetivo el construir un equipo computacional que lograra el poder de procesamiento de 100 computadoras. IBM no fue el primer pionero en querer crear esta supercomputadora, en el mismo año Livermore Automatic Research Computer, aparece como competencia ante esta apuesta que hace IBM de investigar para desarrollar esta supercomputadora, finalmente tres años mas tarde se concluyen estos proyectos que dan como resultado los dos primeros productos de supercomputadoras conocidas como: STRETCH y LARC.

Debido a estas dos iniciativas, se empiezan a producir muchas mas supercomputadoras en el mercado, teniendo diferencias y variaciones en cuanto a arquitecturas, así como tipos de Software que implementaban. La principal razón para construir estas maquinas paralelas fue la de disminuir el tiempo total de la ejecución de una aplicación, resolver en un menor tiempo problemas complejos y permitir la ejecución simultánea de diversas tareas.

II. MARCO TEÓRICO

En la siguiente sección se mostrará un marco teórico de términos y técnicas generales incluidas en la programación en paralelo.

Clúster de computadoras

El desarrollo de sistemas operativos y compiladores del dominio público (Linux y GNU software), estándares para el paso de mensajes (MPI) [4] y conexiones universales a periféricos (PCI), han hecho factible económicamente el acceso a recursos computacionales de producción masiva (CPU’s, discos, redes).
La principal desventaja que presentan los proveedores de multiprocesadores es que deben satisfacer una amplia gama de usuarios, es decir, deben ser de propósito general. Esto aumenta los costos de diseño y producción de equipos, así como los costos de desarrollo del software que va con ellos: sistema operativo, compiladores y aplicaciones. Todos estos costos deben ser añadidos cuando se hace una venta.

Por supuesto, alguien que solo necesita procesadores y un mecanismo de paso de mensajes no debería pagar por todos estos añadidos que nunca usará. Estos usuarios son los que están impulsando el uso de clústers de computadoras.

Ventajas del uso de clústers de computadoras para procesamiento paralelo:

• La explosión masiva de redes implica que la mayoría de los componentes necesarios para construir un clúster son vendidos en altos volúmenes y a bajo costo, y por lo tanto tener acceso a sus componentes es sencillo y económico. Ahorros adicionales se pueden obtener debido a que solo se necesitaría una tarjeta de vídeo, un monitor y un teclado por clúster. El mercado de los multiprocesadores es más reducido y económicamente más costoso.

• Remplazar un componente defectuoso en un clúster es relativamente trivial comparado con hacerlo en un multiprocesador, permitiendo una mayor disponibilidad en clústers cuidadosamente diseñados. [1]

Algoritmos en paralelo

Al momento de diseñar algoritmos paralelos, existe un conjunto de paradigmas de programación que pueden ser usados. Estos paradigmas permiten guiar el desarrollo eficiente del código paralelo, algunos de los cuales incluso, ya han sido usados con éxito en el desarrollo de código secuencial. Los paradigmas no son algoritmos, son estrategias de resolución de problemas. El propósito de los paradigmas es dos:

• Tener ideas que permitan resolver problemas diferentes por medios similares,

• Permitir desarrollar herramientas de diseño que ayuden a los programadores. [6]

Para el caso de los paradigmas de programación paralela, estos encapsulan información sobre los patrones de comunicación y el esquema de descomposición de tareas. Existen diversos paradigmas de ellos, a continuación, se muestran algunos:

• Maestro / Esclavo: consiste en dividir el trabajo en subtareas que serán realizadas por los diferentes esclavos bajo control del maestro. Esquemas MIMD (Multiple instruction stream, multiple data stream) con Memoria Compartida trabajan perfectamente bajo este enfoque. [5]

• Divide y Vencerás: es un esquema en que la resolución del problema se descompone en varios subproblemas, los cuales a su vez se subdividen en subproblemas, hasta llegar a un nivel donde los subproblemas no pueden (o deben) ser más divididos. Cuando los subproblemas son resueltos, ellos regresan sus resultados a sus niveles superiores. Cada subproblema puede ser resuelto independientemente y sus resultados combinados dan el resultado final. En computación paralela, dichos subproblemas pueden ser resueltos al mismo tiempo, si se tienen suficientes procesadores disponibles.

• Árbol Binario: es un caso particular del paradigma anterior, tal que los cálculos sobre N datos (n1, …, nN) son descompuestos en cálculos sobre (n1, …, nN/2) y (nN/2+1, ..., nN) datos. Por ejemplo, para sumar N enteros (n1, …, nN), donde N es múltiplo de 2, y si colocamos los datos en las hojas del árbol, dicho cálculo podría hacerse de la siguiente forma: N/2 procesadores son empleados para calcular la suma de pares (n1, n2), …, (nN-1, nN) en un solo paso. Después, N/4 procesadores ejecutan la misma tarea para los pares de elementos de datos recién calculados, y así sucesivamente. El cálculo procede desde las hojas a la raíz del árbol. [2]

Arquitectura de Von Neumann

En 1944, John von Neumann se incorporó al proyecto ENIAC. El grupo quería mejorar la forma en que se introducían los programas y pensaron en almacenar los programas como números; von Neumann ayudó a cristalizar las ideas y escribió un memorándum proponiendo un computador de programa almacenado denominado EDVAC (Electronic Discrete Variable Automatic Computer). Herman Goldstine distribuyó el memorándum y le puso el nombre de von Neumann, a pesar de la consternación de Eckert y Mauchly, cuyos nombres se omitieron. Este memorándum ha servido como base para el término comúnmente utilizado computador Von Neumann. [3]

Se caracterizaba por guardar las instrucciones de los procesos y los datos en una memoria electrónica, a diferencia de cómo se modelaban los computadores de la época a través de una conexión de cables. Sus Componentes principales:

- Memoria
- Unidad de control
- Unidad Aritmética Lógica
- Entradas/Salidas

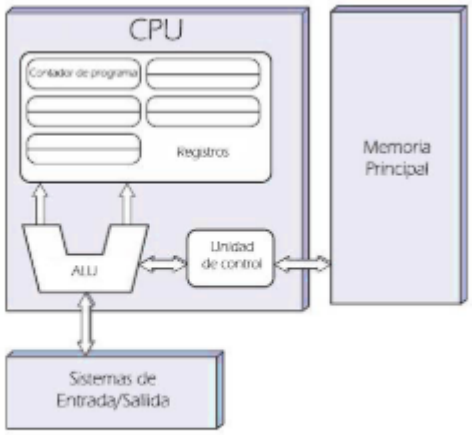


Figura 1. Modelo de la Arquitectura Von Neumann

III. MATERIAL

Para la prueba de rendimiento que se mencionará mas adelante se trabajó en equipos con las siguientes características:

PC:
Procesador Intel I3 de 4 núcleos, disco duro 1TB, memoria RAM 6GB
Clúster:
1 Nodo Maestro.
Procesador Intel I3 de 4 núcleos, disco duro 1TB, memoria RAM 6GB, 2 tarjetas de red 1 Gbps.
16 Nodos esclavo con las mismas características
1 switch Cisco SG2600-26 Gigabit Ethernet

IV. PRUEBAS

Por medio de un programa creado en lenguaje C y a través de la implementación de las librerías MPI y Open MPI propias de la arquitectura de clúster de computadoras se ejecutó un programa en los equipos descritos anteriormente dentro del apartado de materiales para obtener todos los números primos existentes dentro de un intervalo dado. Para la ejecución de este ejemplo el intervalo dado fue: 1 X 10 000000.

Código utilizado en el programa [4]

```
#include <stdio.h>
#include <mpi.h>
#include <math.h>

/**
*Verifica si un número es primo o no
*/
int es_primo (int n) {
        //función que determina si un número dado
es primo o no
        int j, ret =  1;
        for(j=2; j<(int)sqrt(n); j++)
                if( n%j == 0) ret = 0;
                        //
        return ret;
}
 /***Main */

int main(int argc, char **argv) {

        int rango;
        int tamanio;

        // inicializa valores MPI del cluster
        MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rango);
        MPI_Comm_size(MPI_COMM_WORLD,&-
tamanio);

        int n2;
        int primosp=0;  // almacena número de  pri-
mos parciales
        int primostot; //  almacena número de  pri-
mos totales
        int iterac = 10000000;
        int n1 = rango * iterac; // rango de iteración

 while(1) {
                n2 = n1 + iterac;
                if(n1 == 0 ) n1 = 1;

                // analiza rango en cada nodo del clús-
ter
                int n;

                for(n = n1; n < n2;  n++) {
```

```
if(es_primo(n++))
                                primosp++;
            }
            // envia resultado de cada nodo esclavo al nodo master

            MPI_Reduce(&primosp, &primostot, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

            // prepara el siguiente rango a analizar
            n1 += tamanio * iterac;

            // imprime resultados parciales
            if (!rango)
                  printf("%d primostot <  %d\n", primostot, n1);
      }
      MPI_Finalize();

      return 0;
}
```

V. RESULTADOS

En esta prueba con la ejecución del programa para el intervalo dado, se realizó de manera inicial en una PC con las características anteriormente mencionadas, posteriormente para comprobar la eficiencia del clúster se realizó la ejecución del programa con el mismo intervalo de números y como resultado se obtuvieron los siguientes valores mostrados a continuación en la tabla 1:

Nodos	Núcleo	Tiempo de procesamiento
1 PC	4	5.078
17 clúster	68	0.013

El desarrollo del experimento 1 se realizó en una PC con sistema operativo Linux CentOS 7 y las librerías necesarias para compilar y ejecutar código en C; la prueba arrojó la impresión de un listado, el cual es un rango determinado de números (ingresado por el usuario) dentro del cual el código arroja la cantidad de números primos existentes dentro de dicho rango, y continúa con siguiente rango sucesivamente hasta llegar al final de la iteración.

El desarrollo del experimento 2 se realizó en el nodo maestro del clúster el cuál cuenta con sistema operativo Linux CentOS 7, librerías necesarias para compilar y ejecutar código en C, librerías MPI y Open MPI necesarias para ejecución de programas en paralelo , además de la carpeta share dentro de la cual se realiza el paso de mensajes entre los nodos; la prueba arrojó la impresión de un listado, el cual es un rango determinado de números (ingresado por el usuario) dentro del cual el código arroja la cantidad de números primos existentes dentro de dicho rango, y continúa al siguiente rango, con la diferencia de que algunas ocasiones no ocurrió sucesivamente, debido a que algunos nodos enviaron su información mas rápido y el listado se imprimió en pantalla de manera intercalada (en desorden) hasta llegar al final de la iteración.

VI. CONCLUSIÓN

Este trabajo de investigación plantea la comparación del tiempo de ejecución de un código en lenguaje C en dos arquitecturas diferentes, misma que es abordada en el apartado 3 de Resultados por medio de 2 experimentos, a través de los cuales se puede demostrar que la arquitectura del clúster de alto rendimiento (HPC) implementada en el Laboratorio de la Maestría en Sistemas Computacionales del Instituto Tecnológico de Acapulco, México; para esta comparación fue cerca del 97% mas rápida que la arquitectura de una computadora personal con las mismas características de software. Como trabajo a futuro se planea ocupar el clúster como herramienta para realizar criptoanálisis al archivo SAM de Windows y encontrar vulnerabilidades. Actualmente existe la herramienta OpenSSL, en la que se implementan diferentes tipos de algoritmos criptográficos incluyendo el AES en una forma mas eficiente, más adelante podrá ocupar dicha herramienta para hacer otro tipo de análisis de rendimiento al algoritmo AES y otros algoritmos mas de cifrado simétrico como asimétrico.

V. BIBLIOGRAFÍA

Hwang, K., y Briggs, F. Computer Architecture and Parallel Processing, McGraw-Hill, New York, 1984.

J. Aguilar, E. Leiss. Introducción a la computación Paralela. Mérida, Venezuela, Graficas Quintero 1ra Edición, 2004.

John I. Hennessy, David A. Patterson. Arquitectura de computadores. Un enfoque cuantitativo. México, México, Mc Graw Hill, 2002.

The Message Passing Interface (MPI) standard (2012). https://www.mcs.anl.gov/research/projects/mpi/index.html

Snir, M., Dongarra, J., y otors. MPI: The Complete Reference. MIT Press, Massachusetts, 1996.

Foster, I. Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, Addison-Wesley, New York, 1995.