# Procesamiento HPC en Sistema CRIARD

Matías Adagio, Erik Álvarez, Beatriz Yanina Caballero, Valeria Couto, Santiago Díaz

Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina
mati.adagio@gmail.com, alvarez.a.erik@gmail.com, caballero.yani@gmail.com,
vale.alejandra1409@gmail.com, santiago.d.diaz@gmail.com

**Resumen.** En esta investigación se intenta comprobar si es posible realizar el procesamiento de un gran volumen de datos generado en forma simultánea por el sistema CRIARD (Cuna Inteligente) instalado en numerosos hospitales, utilizando paralelismo mediante OpenMPI.

**Palabras claves:** HPC, OpenMPI, Cuna, Bebé

## 1 Introducción

CRIARD es un sistema embebido que permite monitorear el descanso de un bebé, dando a los padres la posibilidad de conocer el estado en que se encuentra su hijo sin que ellos estén presentes. También permite controlar la luminosidad de la habitación en la que se encuentra, emitir sonidos relajantes y mecer la cuna a través de una aplicación Android conectada al sistema a través de Bluetooth.

Hoy en día existen diversos sistemas que permiten monitorear el estado de salud de las personas en tiempo real o que posibilitan el monitoreo de los estados del sueño para determinar distintas patologías. Todos estos aspectos pueden incluirse dentro de nuestra aplicación en un futuro.

El propósito de esta investigación es conocer la forma en que se puede resolver el procesamiento de información arrojada por los distintos sensores en lugares donde nuestro sistema embebido pueda usarse en forma masiva. Esta situación podría generarse cuando CRIARD sea utilizado en una gran cantidad de hospitales y se necesite conocer la situación general dentro de cada centro de salud y, a mayor escala, obtener datos a nivel provincial y/o nacional.

Con el fin de mejorar el acceso a cada sistema embebido, se propone utilizar OpenMPI ya que se utiliza memoria distribuida, es thread-safe y open source.

## 2 Desarrollo

Cada centro de salud dispondrá de un servidor que manejará los datos arrojados por los distintas cunas en forma independiente, por lo que podemos decir que se trabajará en entornos con memoria distribuida. A su vez, nuestro sistema embebido es monitoreado a través de conexión Bluetooth por un dispositivo Android recibiendo el muestreo de los diferentes sensores (luminosidad, humedad, signos vitales) calculando constantemente un promedio de estos valores. A partir de los datos obtenidos, se activarán las alertas correspondientes a cada uno de los responsables del centro de salud.
Debido a que en un hospital se podrán utilizar alrededor de 100 cunas cada una de los cuales tendrán los sensores detallados a continuación:

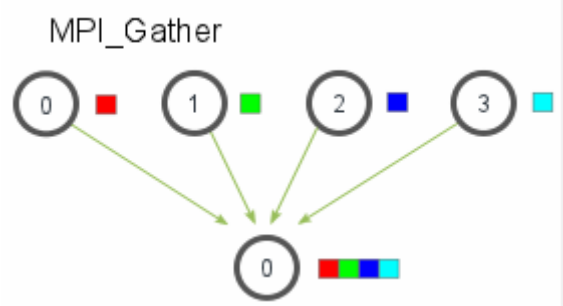
- 2 Sensores de Humedad ambiental (uno dentro de la cuna y otro fuera)
- 2 Sensores de Temperatura ambiental (uno dentro de la cuna y otro fuera)
- Sensor de agua en el colchón (Indica si el colchón está mojado)
- Sensor de rayos UV para monitorear la exposición a lámparas
- Sensor de vibración para detectar movimientos anormales del bebé
- Sensor de temperatura corporal
- Sensor de signos vitales
- 2 Sensores de ruido (uno para detectar sonidos relacionados al bebé y otro para ruidos ambientales)

Para garantizar la precisión de estos datos, se desea hacer uso de la API OpenMPI para poder utilizar el paralelismo en nuestro sistema. Se ejecutarán los procesos de cada cuna en paralelo y cada una de ellas se comunicará por ssh. El tipo de mensajería colectiva a utilizar será Gather, donde la computadora nodo central recibirá los datos de cada cuna para poder procesarlos y calcular estadísticas que permitan conocer el estado general de los pacientes del centro de salud y determinar distintas patologías. Luego, esta información será enviada, a través del tipo de mensajería Scatter, a cada dispositivo Android conectado donde se podrán consultar los resultados obtenidos permitiendo (en el caso de los responsables de los distintos centros de salud) activar las medidas de prevención necesarias.
Nos centramos en el desarrollo de metodologías, modelos y construcción de software para también administrar y gestionar el consumo de energía y prestaciones de sistemas de cómputo paralelo. Tenemos como objetivo: Predicción de energía y rendimiento. Es importante proveer a un administrador de sistema de herramientas que permitan predecir la energía y el rendimiento que producirían distintas aplicaciones paralelas, y así poder seleccionar la configuración adecuada que mantenga el compromiso deseado entre tiempo de ejecución y eficiencia energética.
Se utilizará Cloud Computing para escalar la información recolectada por los sistemas embebidos de varios hospitales. Nuestra optimización de performance en la red estará específicamente diseñada para operaciones colectivas MPI.

## 3 Explicación del algoritmo.

El esquema elegido es MPI y es principalmente usado en memoria distribuida, aunque también puede ser utilizado en memoria compartida. La figura muestra el paso de mensajes para la paralelización usando MPI, donde cada proceso MPI es responsable de la evaluación de los valores. Se puede obtener un grado mayor de paralelismo mediante la paralelización de cada proceso MPI.

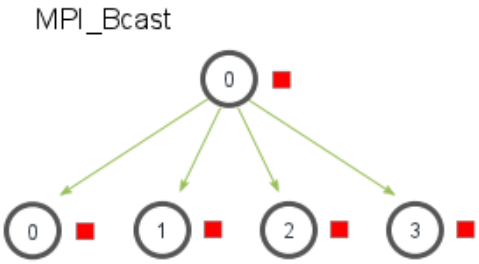
El algoritmo consiste en utilizar el método de paso de mensajes llamado Gather, donde los datos de cada cuna serán enviados a un nodo central, el cual recibe y procesa dichos datos. También se utilizará el tipo de mensajería Broadcast para comunicar los resultados obtenidos a todos los dispositivos Android conectados.
En primer lugar, los datos a enviar en forma paralela serán los obtenidos desde los sensores de luminosidad, humedad y signos vitales.




Los elementos de cada proceso son ordenados por su número. La función MPI_Gather es la siguiente:

```
MPI_Gather(
    void* send_data,
    int send_count,
    MPI_Datatype send_datatype,
    void* recv_data,
    int recv_count,
    MPI_Datatype recv_datatype,
    int root,
    MPI_Comm communicator)
```

Luego, los datos obtenidos serán procesados calculando las valores máximo, mínimo y promedio de cada cuna. Con esta información, se procederá a escalar la información a nivel centro de salud y se cotejará con las fichas médicas de cada uno de los pacientes para detectar posibles síntomas, y llegado el caso un brote de una infección nosocomial o intrahospitalaria.
Esto, permitirá mejorar el cuidado de cada uno de los niños notificando los resultados a los profesionales de la salud a través de dispositivos Android de fácil transporte.




La función MPI_Bcast es la siguiente:

```
MPI_Bcast(
    void* data,
    int count,
    MPI_Datatype datatype,
    int root,
    MPI_Comm communicator)
```

El algoritmo a implementar la comunicación entre el nodo raíz y las cunas que envían datos es el siguiente:

```
#include <mpi.h>

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

        /* Hasta este momento no hemos utilizado
        MPI en nuestro programa */
        MPI_Init(&argc,&argv);
        .
        /* Aquí uso las llamadas a la librería MPI */

        /* Averiguamos el rango de nuestro proceso */
        MPI_Comm_rank(MPI_COMM_WORLD, &mi_rango);

        /* Averiguamos el número de procesos que estan
        * ejecutando nuestro programa
        */
        MPI_Comm_size(MPI_COMM_WORLD, &p);

        if (mi_rango != 0) {
        /* Obtengo datos de los sensores */
        datos = obtener_datos(...);
                /* Envío datos al nodo raíz */
                MPI_Gather(…);

        } else {
        /* mi_rango == 0 entones ejecuto código para el nodo raíz*/
        for (fuente = 1; fuente < p; fuente++) {
        /* Por cada cuna, recibo los datos que fueron enviados */
                MPI_Gatherv(…);
                procesar_datos(...);
        }
        /* Envío los resultados a cada uno de los dispositivos */
        MPI_Bcast(…);
    }
  }

  /* Termina con MPI. Recordemos que después de
   * esta llamada no podemos llamar a funciones
   * MPI, ni siquiera de nuevo a MPI_Init
   */
    MPI_Finalize();
}
```

## 4 Pruebas que pueden realizarse

Una de las primeras que debe realizarse es la ejecución en forma secuencial y luego, ejecutar el sistema con paralelismo. De esta forma se puede verificar si realmente existe una mejora al aplicar OpenMPI.
Otras de las pruebas a llevar a cabo será la medición de los distintos sensores cuando hay pocos niños utilizando la cuna (por ejemplo 10 niños). Luego, realizar la prueba involucrando a todos los sistemas embebidos que se encuentran en el hospital para comprobar que el sistema funcione correctamente cuando se desee procesar gran cantidad de información.

## 5 Conclusiones

Durante esta investigación se aborda la problemática del seguimiento de datos de los bebés mientras están en su cuna inteligente. Aplicado a una gran cantidad de cunas, podemos procesar paralelamente la cantidad de datos que cada una puede enviar a partir de sus sensores, lo que permitiría poder generar estadísticas y prevenir distintas patologías con el sensor de signos vitales.

Implementar OpenMPI nos permite procesar gran cantidad de datos y enviarlos a través del paso de mensajes, aprovechando el máximo procesamiento de cada sistema embebido paralelamente. Cuando el tamaño de los datos es moderado y el problema es la computación intensiva en paralelo para procesar los datos de los sensores de signos vitales, MPI es el framework que puede ser considerado.
MPI es una buena opción cuando los procesos necesitan ser ejecutados en paralelo y distribuidos con coordinación entre muchos procesos.

Para un próximo trabajo, una buena opción sería implementar un nodo raíz en varios hospitales utilizando OpenMPI, obteniendo estadísticas a mayor escala.

## 6 Referencias


1. Javier Balladini, Marina Morán.: Aplicaciones de Cómputo Intensivo con Impacto Social http://sedici.unlp.edu.ar/bitstream/handle/10915/68232/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y - Página 3 (2018)
2. Juan F. R. Herrera, Eligius M. T. Hendrix, Leocadio G. Casado y René Haijema: Paralelismo de datos en la obtenci´on de Tablas de Control de Tráfico con informaci´on de llegada https://riuma.uma.es/xmlui/bitstream/handle/10630/8093/jpher-jpar-leo.pdf?sequence=1&isAllowed=y - Página 5 (2014)
3. Sol Ji Kang, Sang Yeon Lee, and Keon Myung Lee: Performance Comparison of OpenMP, MPI, and MapReduce in Practical Problems https://www.researchgate.net/publication/282538300_Performance_Comparison_of_OpenMP_MPI_and_MapReduce_in_Practical_Problems - Página 8 (2015)
4. Yifan Gong, Bingsheng He, Jianlong Zhong: Network Performance Aware MPI Collective Communication Operations in the Cloud – Página 2 (2015) http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.728.2038&rep=rep1&type=pdf
5. Examples using MPI_GATHER, MPI_GATHERV - https://www.mpi-forum.org/docs/mpi-1.1/mpi-11-html/node70.html
6. Standard Gather - https://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-1.1/node69.htm#Node69