Procesamiento HPC en Sistema de control de aire accondicionado

Bedetti Nicolas, Hoz Aylen, Torres Quimey

Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina
nbedetti@gmail.com; ailu.hoz28@gmail.com; quimey.torres@gmail.com

Resumen. El objetivo del presente documento de investigación es describir la opción por la cual se ha optado para manejar el gran volumen de datos entregados por diversos embebidos "SmartAir" dispersos en un edificio, teniendo en cuenta que el analisis de los datos se debe realizar de forma concurrente y la respuesta a los mismos debe ser en tiempo real como requisito. En base a este problema planteado se propone utilizar el paradigma OpenMP debido al soporte de paralelismo que este brinda.

Palabras claves: HPC, OpenMP, Servidor.

1 Introducción

La investigación desarrollada en este documento se aplica al sistema embebido "SmartAir", sistema de control de aire accondicionado inteligente. Este embebido recibe información principalmente de los sensores de temperatura, uno interno para controlar la temperatura del aire que sale del dispositivo y uno externo que mide la temperatura ambiente, ademas de un sensor de inclinación que permite informar cuando el dispositivo es maltratado debido a que es portable; con esta información, según el modo elegido (frio, calor o automático (24°C)), los actuadores que lo conforman responderán a las peticiones de cada modo.

Se plantea el uso de diversos embebidos en un edificio aplicado a una industria o edificio en el cual el control de la temperatura ambiente es vital y deben realizarse mediciones constantes en tiempo real, ademas de procesar esta información para enviar reportes constantes de los usuarios del sistema que monitoreán y controlan los dispositivos desde una aplicación móvil. De tal forma, se presenta el problema del gran volumen de datos que debe procesar simultáneamente un servidor central en la nube encargado del control de todos los sensores y actuadores presentes en cada habitación del edifico, lo cual podrían generar que se sature y colapse.

Se propone procesar estos grandes volumenes de información de forma paralela, de tal forma se mejora el control por parte del servidor cloud y mejoran los tiempos de respuesta a las peticiones. Debido a los beneficios que otorga la implementación de la

plataforma OpenMP debido a que no solo brinda paralelismo, sino que es portable, seguro y simple de implementar.

Se plantea la aplicación de una arquitectura FireBase [1] con un servidor cloud al cual se conectan los embebidos que envían constantemente las mediciones de los sensores para procesar la información para reportes y ser enviada a los dispositivos móviles que se encuentran suscriptos a estos, ademas el servidor deberá responder a las peticiones de los usuarios desde la aplicación, lo cual se asemeja a la utilización en sistemas SCADA (Supervisory Control And Data Acquisition) que utiliza técnicas de HPC con una gran computadora o servidor como esclavo en la red de los sensores y actuadores para la respuesta en tiempo real de peticiones y toma de mediciones [2].

2 Desarrollo

La presente investigación tiene como base la conexión de múltiples embebidos "SmartAir" aplicados al dominio de un edificio o industria. Actualmente, el embebido se trata de un aire accondicionado portatil con un sistema de control de la temperatura. Como se planteo en el inicio todos los embebidos serán conectados a un servidor central, por lo tanto, el análisis de la información recibida en tiempo real sería centralizada y controlada por un único servidor cloud, lo cual generaría, como ya se mencionó anteriormente, un retardo en los tiempos de respuesta de la aplicación para los usuarios de los dispositivos.

Para el desarrollo propuesto anteriormente es necesario contar con un sistema multiprocesador para la paralelización de los distintos procesos que van a responder a las peticiones de los usuarios. OpenMP permite inciar a partir de un código en serie y provee un enfoque incremental al paralelismo.

Para esta aplicación, se contará con un proceso inicial que empezara la ejecución, que se conoce como región serie, y luego se utilizarán distintos algoritmos que permitirán encolar a las peticiones de los usuarios, para luego ser distribuidas entre distintos hilos a través de directivas como fork/join mencionadas anteriormente, formando la región paralela [3]. También es necesario contar con una base de datos que pueda trabajar de forma paralela para almacenar la información obtenidad de los embebidos en logs para los reportes de tiempo real.

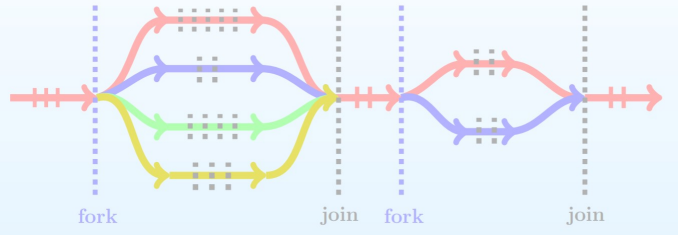

Fig 1: Xiaoxu Guan: Introduction to OpenMP (2016) pag 6. [3]

3 Explicación del algoritmo.

Cada embebido disperso en el edificio cuenta con dos sensores de temperatura (uno interno y otro externo) y un sensor de inclinación que permite apagar el dispositivo para que no se dañe al ser transportado durante su funcionamiento (además de brindar una alarma sonora con un buzzer), por otro lado, se recibe la velocidad a la cual funcionan los ventiladores que envían el aire hacia el exterior del dispositivo. Todos estos datos de entrada serán enviados por el embebido en tiempo real al servidor cloud y serán utilizados para el control de la temperatura ambiente de la habitación en la cual se encuentra el dispositivo según las preferencias del usuario que lo controla, teniendo encuentra la seguridad de los componentes internos del mismo.

El servidor central recibirá estos datos generando e informará a los usuarios en tiempo real el cambio en las mediciones identificando los embebidos, los valores obtenidos de los sensores, el modo en el que se encuentra y fecha y hora, lo cual también se almacenará en un log. El usuario de la aplicación podrá tener un análisis o reporte en tiempo real del maltrato que reciben los equipos, cuanto tiempo son utilizados, la energía que gastan (según el tiempo que están activos) y cuan eficientes son en enfriar o calentar la habitación en la que se encuentran.

Como se comentó en un principio, se utilizará una base de datos central situada en el servidor cloud (arquitectura FireBase) para el log de acciones realizadas común a todos los embebidos y todos los hilos accederán de forma concurrente, por lo tanto, se implementarán regiones críticas para la escritura de los registros, protegiendo la integridad de la información [4].

Se decidió utilizar un servidor cloud con un procesador de ocho núcleos, por lo que se utilizarán ocho hilos y se implementarán las librerías de OpenMP (#include <omp.h>) para la implementación de concurrencia [5].

```
#include <omp.h>
void main()
{
  //se plantea el uso de dos secciones, una para recibir las mediciones de
  //los sensores en tiempo real y otra para la respuesta a las peticiones
  //de control de los usuarios que llegan desde la aplicación
  //Como se cuenta con 8 nucleo (8 hilos) se le otorgarán 6 a la sección de
  //mediciones que debe recibir la información en tiempo real y 2 a la sección
  //de respuesta a las peticiones de los usuarios.

 #pragma omp sections
 {
    #pragma omp section
    {
      // en la seccion de mediciones en tiempo real se encolarán todas las mediciones
      //de los dispositivos con el idEmbebido, mediciones de los sensores y modo en el
      //que se encuentra en un vector que será recorrido por un for paralelizado
      //suponemos que contamos con 4000 embebidos funcionando en simultáneo
      #pragma omp for
      for (int 1=0, i<4000, i++)
      {
        escribirLog(idEmbebido, lecturaSensores, modo);
        //Se obtienen los dispositivos a los cuales se debe enviar el reporte
        int Suscriptores[] = obtenerDispositivosSuscriptos(idEmbebido);
        //Se genera el reporte
        generarReporte(idEmbebido);
        enviarReporte(&Suscriptores);
      }
    }
    #pragma omp section
    {
      //De la misma forma que se realiso con las mediciones se encolan en un vector
      //las peticiones de los usuario que pueden ser cambio de modo, subir o bajar temperatura
      //o aumentar o disminuir la velocidad de ventiladores.
      for (int 1=0, i<4000, i++)
      {
        escribirLogAcciones(idEmbebido, lecturaAcciones, modo);
        enviarSeñalEmbebido(idEmbebido, lecturaAcciones, modo);
      }
    }
 }
}
```

```
void escribirLog(idEmbebido, lecturaSensores, modo)
{
    //región crítica para escribir en la base de datos el log
    #pragma omp critical (RegistrarEntrada)
    {
      //se guarda en la base el idEmbebido, con fecha y hora y
      //los valores medidos en los sensores y el modo en que se encuentra
    }
}
```

4 Pruebas que pueden realizarse

En primer lugar, se puede realizar un prueba que compare el tiempo de respuesta utilizando OpenMP tomando las mediciones de un solo embebido colocado en una habitación y luego compararla probando en simultáneo todos los embebidos colocados en todas las habitaciones realizando una prueba de exigencia máxima,

verificando que el sistema responda a las peticiones de tiempo real en los tiempos de respuesta esperados.

Por último, se puede realizar una prueba que compare el tiempo de respuesta utilizando OpenMP y sin utilizarlo al utilizar al menos 20 embebidos de forma simultánea.Como resultado de esta prueba veríamos los beneficios que otorga el uso de OpenMP en cuanto a paralelismo y tiempo de respuesta.

5 Conclusiones

Durante el desarrollo de este trabajo se propuso expandir el domino de los embebidos "SmartAir" para controlar varios dispositivos en un edificio utilizando OpenMP, lo cual nos permitió paralelizar el procesamiento de la información y guardarla de forma segura en una base de datos.

La investigación realizada nos permitió aprender a resolver un problema serial de tiempo real de forma paralela.

Por último, a pesar de que la paralelización hizo que los tiempos de respuesta se redujeran considerablemente, la utilización de un servidor cloud puede generar problemas en caso de fallas en el servidor, por lo tanto, en futuras mejoras se implementaría el concepto de computación distribuida en diversos servidores cloud con el uso de MPI.

En conclusión, OpenMP resulto ser la mejor elección para el desarrollo de este trabajo debido a que como se describió en un principio brinda paralelismo, es portable, seguro y simple de implementar.

6 Referencias

1. Caldera Carlos, Lopez Jesús, Olivas Héctor y Gallegos José: Diseño de sistema de control automatizado con sistemas embebidos, aplicaciones móviles y el internet de las cosas (2017) http://www.ecorfan.org/bolivia/researchjournals/Tecnologia_e_innovacion/vol4num11/Revista_de_Tecnologia_e_Innovacion_V4_N11_6.pdf

2. Birman, Kenneth P., Lakshmi Ganesh, and Robbert Van Renesse: Running smart grid control software on cloud computing architectures. (2011) http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.229.4216&rep=rep1&type=pdf

3. Xiaoxu Guan: Introduction to OpenMP (2016) http://www.hpc.lsu.edu/training/weekly-materials/2016-Spring/OpenMP-06Apr2016.pdf

4. Adam Bird, David Long, Geoff Dobson: Implementing Shared Memory Parallelism in MCBEND (2017) https://www.epj-conferences.org/articles/epjconf/pdf/2017/22/epjconf_icrs2017_07042.pdf

5. Chapman Barbara: How OpenMP is Compiled (2015) http://www.compunity.org/events/pastevents/parco07/tut-compilerrev-chapman_new.pdf