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 la aplicación del modelo embebido "SmartAir" a gran escala en un edificio, teniendo en cuenta que el analisis de los datos se debe realizar de forma concurrente 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; 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.

Al aplicar el embebido "SmartAir" a gran escala en un edificio (oficina, escuela, vivienda, etc.) se presenta el problema del gran volumen de datos que debe procesar simultáneamente un servidor, de todos los sensores y actuadores presentes en cada habitación del edifico, esto podrían generar que se sature y colapse, de tal forma los usuarios del sistema notarían retardos en la respuesta a sus peticiones.

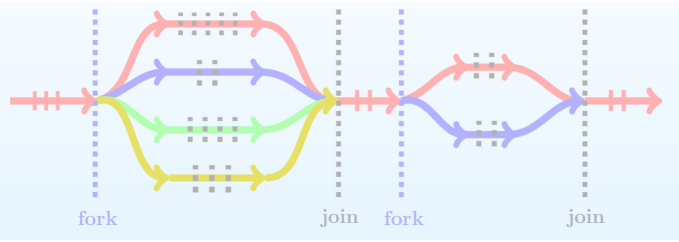
Se propone procesar estos grandes volumenes de información otorgados por los sensores instalados en cada espacio de forma paralela, de tal forma se mejora el control por parte del servidor y mejoran los tiempos de respuesta a las peticiones. Debido a los beneficios que otorga la implementación de la plataforma OpenMP con respecto al paralelismo en comparación con Threads Posix [1], que presenta un mayor esfuerzo en el desarrollo de la protección de memoria y sincronización de hilos, se decidió optar por la primera. La utilización de OpenMP no solo brinda paralelismo, sino que es portable, seguro y simple de implementar.

2 Desarrollo

La presente investigación tiene con base la expansión a gran escala del sistema embebido “SmartAir” al dominio de un edificio. Actualmente, el embebido se trata de un aire accondicionado portatil con un sistema de control de la temperatura de la habitación en la cual es colocado. Debido a la expansión planteada inicialmente, se multiplicaría la cantidad de sensores y actuadores en todas las habitaciones del edificio, por lo tanto, el análisis de la información generada sería centralizada y controlada por un único servidor, lo cual generaría, como ya se mencionó anteriormente, un retardo en los tiempos de respuesta de la aplicación para los usuarios del dispositivo.

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. OpenMP comienza con un solo hilo, pero soporta directivas o pragmas que permiten generar múltiples hilos como el uso de las directivas fork/join, además de permitir cambiar el número de hilos en tiempo de ejecución.

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 futuros reportes.



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), además, se recibe la velocidad a la cual funcionan los ventiladores que envían el aire hacia el exterior del dispositivo. Todos estos datos de entrada son utilizados para controlar 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 un log identificando los embebidos, los valores obtenidos de los sensores, el modo en el que se encuentra y fecha y hora. Toda esta información puede ser utilizada para realizar un análisis o reporte 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 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 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 [2].

```
#include<omp.h>
void main()
{
  //como contamos con 8 hilos vamos a suponer que solo se tienen 8 embebidos,
  //pero en la realidad la cantidad de embebidos debería dividirse en 8.
  int hiloId, cantHilos=8;
  //cada embebido puede leer el valor de dos sensores de temperatura, un sensor
  //de inclinación y la velocidad de un ventilador, por lo tanto, tendrá
  //4 valores de lectura, luego tendrá un identificador de embebido (numérico,
  // el modo en el que se encuentra
  //(encendido (frio (1), calor (2), automático (3), seguro (4)) o apagado (5))
  // y por último el valor de temperatura deseado y la velocidad del ventilador.
  float lecturaSensores[4];
  int idEmbebido, modo;
  //inicio de paralelización
  #pragma omp parallel private (hiloId, idEmbebido, modo, lecturaSensores)
  while (true)
  {
    hiloId = omp_get_thread_num();
    //obtenemos el n° de hilo que coincide con el n° de embebido
    switch(hiloID)
    {
      case 0:
        idEmbebido=1;
        break;
      case 1:
        idEmbebido=2;
        break;
      .
      .
      .
    }
    obtenerMedicion(idEmbebido, &lecturaSensores, modo);
    responderAccion(idEmbebido, &lecturaSensores, modo);
  }
}
```

```
  void obtenerMedicion(idEmbebido, lecturaSensores, modo)
  {
    //obtengo las mediciones y el modo en que se encuentra
    //el embebido para poder responder a las acciones.
    //escribimos el log en la base con las mediciones iniciales tomadas.
    escribirLog(idEmbebido, &lecturaSensores, modo);

  }
  void responderAccion(idEmbebido, lecturaSensores, modo)
  {
    //según el modo se realizarán determinadas acciones
    if (modo == 1) //modo frio
    {
      ledAzul(idEmbebido, ON);
      ledRojo(idEmbebido, OFF);
      ledVerde(idEmbebido, ON);
      señalSonora(idEmbebido); //hacer sonar el buzzer
      bajarTemperatura(idEmbebido, lecturaSensores);
    }
    else if(modo == 2) //modo calor
    {
      ledAzul(idEmbebido, OFF);
      ledRojo(idEmbebido, ON);
      ledVerde(idEmbebido, ON);
      señalSonora(idEmbebido); //hacer sonar el buzzer
      subirTemperatura(idEmbebido, lecturaSensores);
    }


    else if(modo == 2) //modo calor
    {
      ledAzul(idEmbebido, OFF);
      ledRojo(idEmbebido, ON);
      ledVerde(idEmbebido, ON);
      señalSonora(idEmbebido); //hacer sonar el buzzer
      subirTemperatura(idEmbebido, lecturaSensores);
    }
    else if(modo == 3) //modo automático
    {
      ledAzul(idEmbebido, ON);
      ledRojo(idEmbebido, ON);
      ledVerde(idEmbebido, ON);
      señalSonora(idEmbebido); //hacer sonar el buzzer
      //Si la temperatura externa es más alta que 24°C
      bajarTemperatura(idEmbebido, lecturaSensores);
      //si la temperatura externa es menor a 24°C
      subirTemperatura(idEmbebido, lecturaSensores);
    }else if(modo == 4) //modo seguro, se detecto sensor de inclinación
    {
      ledAzul(idEmbebido, ON);
      ledRojo(idEmbebido, ON);
      ledVerde(idEmbebido, ON);
      señalSonora(idEmbebido); //hacer sonar el buzzer
      Peltier(idEmbebido, OFF);
      Ventilador(idEmbebido, OFF);
    }else if(modo == 5) //modo apagado
    {
      ledAzul(idEmbebido, OFF);
      ledRojo(idEmbebido, OFF);
      ledVerde(idEmbebido, OFF);
      Peltier(idEmbebido, OFF);
      Ventilador(idEmbebido, OFF);
    }
  }

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.

En segundo lugar, 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.

Por último, se realizaría una prueba de máxima exigencia con todos los embebidos del edificio solicitando peticiones en simultáneo, para verificar que el sistema responda en los tiempos de respuesta esperados.

5 Conclusiones

<<Las concluciones del trabajo deben tener:>>

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 forma paralela, debido a que el mismo código se repetiría en todos los embebidos, por lo tanto, era mejor paralelizarlo.

Por último, a pesar de que la paralelización hizo que los tiempos de respuesta se redujeran considerablemente, al registrar el log en una base de datos se genera un cuello de botella, por lo tanto, en futuras mejoras se implementaría el uso de un disco de estado sólido o se mantendrían los logs en memoria temporalmente hasta que cuando sea posible se guarden en la base de datos para futuros análisis.

En conclusión, OpenMP resulto ser la mejor opción a elegir 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. Ting-Hsuan Chien, Jhe-Wei Lin, Rong-Guey Chang: Parallel Collision Detection with OpenMP (2018) https://doi.org/10.1088/1742-6596/1069/1/012180
2. Chapman Barbara: How OpenMP is Compiled (2015) http://www.compunity.org/events/pastevents/parco07/tut-compilerrev-chapman_new.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