Procesamiento HPC en Sistema de automatizacion de luces

Molina Mariano, Sanchez Julian

[1]Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina
Marianoo.molina@gmail.com; jlysanchez@gmail.com

Resumen. Esta investigación se realiza con el fin de optar por una solucion para el procesamiento de los diversos datos entregados por diversos embebidos, dispersos en una casa, siendo un requisito que el analisis se realice de manera concurrente. Para esta aplicación se propone la utilizacion de OpenMP para hacer beneficio del paralelismo.

Palabras claves: OpenMP, HPC, Servidor.

1 Introducción

La investigación que se desarrolla en este documento esta aplicada al sistema LumusSystem, un sistema de iluminación inteligente, en conjunto con una alarma programable, que integra a todo un espacio. Este sistema recibe informacion de los diferentes sensores que cuenta cada espacio, siendo estos: sensor de luz, sensor de sonido, sensor de movimiento en las entradas al cuarto. Como tambien con distintos actuares, para poder encender y apagar las luces, y programar la alarma. Dependiendo del nivel de luz con el que cuenta el ambiente, el sistema determina la necesidad de iluminar el espacio de manera artificial. Asi tambien si detecta movimiento a traves de los sensores PIR encendera las luces.

La problemática que se presenta es la del gran volumen de sensores que contiene una casa con múltiples habitaciones, por lo que un servidor, encargado del control general, podría saturarse o colapsar, haciendo que los tiempos de respuesta no sean los estipulados, como también podría realizar un incorrecto control sobre la iluminación de los distintos cuartos administrados.

Lo propuesto es hacer uso del paralelismo para el procesamiento de la información otorgada por cada espacio (lectura de los sensores), así lograr un correcto control por parte del servidor y mejorar los tiempos de respuesta de este. Esto se puede lograr implementando la plataforma de paralelismo OpenMP, el cual nos otorga portabilidad, simplicidad a la hora de implementar, y seguridad entre hilos. Otra implementación posible hubiese sido POSIX Threads, lo cual sería mas eficiente [4], pero debido a su control de hilos de bajo nivel, sería necesario un riguroso desarrollo sobre la sincronización de hilos y la protección de memoria entre ellos. [1], [2].

2 Desarrollo

Nuestra investigación tiene como origen el despliegue de nuestro sistema LumusSystem, a un dominio más amplio. Actualmente el sistema realiza el control de la luminosidad de una sola habitación, teniendo como entrada las señales emitidas por sensores de luz, movimiento, sonido, en conjunto con un sistema de alarma programable. Debido a que el fin de esta investigación seria la implementación del control de múltiples cuartos, siendo así el análisis de cada habitación lo realizara un servidor común a todos los sensores y espacios; esto generaría un constante flujo de

datos hacia el servidor, lo cual, ya observado anteriormente, afectaría los tiempos de respuesta de este como también fallos en el control del nivel de luz de cada cuarto.

Debido al problemática secuencial se decidió por hacer uso del paralelismo con OpenMP. Transformar un programa secuencial en subprogramas que se ejecutan de manera paralela no es tarea fácil. Al realizarlo uno tiene que ser riguroso a la hora de elegir que funciones son independiente de los resultados de otras, como también que se pierde la facilidad de poder hacer un trackeo de un problema, siempre y cuando este surja [5]. Se vuelve más complicado poder depurar el programa, y una mala implementación de la concurrencia podría hasta generar aumentos en los tiempos de respuesta. Otro factor para tener en cuenta a la hora de desarrollar es el impacto que tendrá las distintas directivas sobre el desempeño y tiempo de ejecución del programa [6], ya que un mal uso de las diferentes directivas con las que cuenta la librería de OpenMP, podría resultar inviable utilizar procesadores de mayor cantidad de Cores.

3 Explicación del algoritmo.

Los valores de lectura de cada embebido son respecto al nivel de luz que recibe el ambiente desde el exterior, como también si se registra el ingreso de una persona, además de contar con una alarma programable y un detector de sonido para activar y desactivar las luces. En base a los distintos valores que puede recibir de un embebido el sistema realiza múltiples acciones, como accionar la persiana, encender o apagar las luces, o hacer sonar la alarma. Si en algún momento de la ejecución ocurre algún error el sistema lo notificara a través de un archivo de log, indicando además del error, sobre que embebido ocurrió.

Asimismo, el servidor llevará un registro de los datos que se fueron generando como también de las acciones realizadas. Esta información podrá ser utilizada para un análisis del consumo de electricidad para la iluminación de la casa, como también de cada espacio.

Al ser los archivos de log y de acciones realizadas comunes a todos los hilos, se implementó regiones críticas para la escritura de estos, protegiendo su integridad. [3]

El servidor cuenta con un procesador de 4 Cores, por lo que se utilizaran cuatro hilos, empleando las librerías de OpenMP para su implementación.

```
#include <omp.h>

void main(){
    //Suponiendo que en este caso se cuenta con cuatro habitaciones automatizadas
    int th_id, nthreads=4;
    //Valores que envian los distintos embebidos en cada habitacion
    valoresLectura[4];
    int[4] acciones ;
    string embebido;
    string error = null;
    string log = "Direccion del log en el servidor"
    string registroDeAcciones = "Direccion del archivo de registro"
    //inicio de parte paralelizada
    #pragma omp parallel private(th_id , valoresLectura, embebido, error, acciones)
    while(true){
        th_id = omp_get_thread_num();
        //Dependiendo de que hilo se esta ejecutando obtengo los valores del embebido que corresponda
        switch(th_id){
            case 0:
                embebido = "direccion del embebido uno";
                break;
            case 1:
                embebido = "direccion del embebido dos";
                break;
            case 2:
                embebido = "direccion del embebido tres";
                break;
            case 3:
                embebido = "direccion del embebido cuatro";
                break;
        }
        acciones = {0,0,0,0};
        obtenerValores(embebido, &valoresLectura, &error);
        analizarValores(&valoresLectura, &embebido, &acciones, &error);
        if(error != null)
            informarError(&error, &embebido);
    }
}
```

```
void obtenerValores( embebido, valoresLectura,  error){
    //obtengo los valores del embedico correspondiente a la direccion
    // definida segun el hilo
    //Cargo los valores en la variable valoresLectura
    //Si hay algun error lo informo en la variable
    receive(embebido,&valoresLectura,error);
}
void analizarValores( valoresLectura, embebido, acciones, error){
    //analizo los valores obtenidos del embebido
    //determino una accion a realizar (de ser necesario)
    //si hay algun error lo informo en la variable
    if(cambioDeLuz(valoresLectura[0]){
        //El sensor de luz determino la ausencia/presencia de luz exterior
        //Por lo que se deberia cambiar el estado de luz artificial
        // y si cuenta con persianas hacer el despligue/cierre de las mismas
        acciones[0] = 1;
        registrarAccion(&acciones[1], &embebido, &error);
    }
    if(movimiento(valoresLectura[1])){
        //Encender luces por deteccion de movimiento
        acciones[1] = 2;
        registrarAccion(&acciones[1], &embebido, &error);
    }
    if(dobleAplauso(valoresLectura[2])){
        //Encender/apagar las luces
        acciones[2] = 3;
        registrarAccion(&acciones[2], &embebido, &error);
    }
    if(checkalarma(valoresLectura[3])){
        //Hacer sonar la alarma
        acciones[3] = 4 ;
        registrarAccion(&acciones[3], &embebido, &error);
    }
    realizarAccion(&acciones, &error);
}
```

```
void realizarAccion( acciones[], embebido, error){
    //realizo la accion correspondiente en base al valor de la accion
    //si hay algun error lo informo en la variable
    send(acciones, embebido);

}
void registrarAccion(accion, embebido, error){
    DateTime fechaHora = DateTime();
    #pragma omp critical (RegistrarAccion)
    {
        String stringAImprimir = "";
        stringAImprimir = stringAImprimir + "Accion: "  + getAccion(accion); //devuelve string correspondiente al Nº de accion
        stringAImprimir = stringAImprimir + " Realizada sobre el embebido: " + embebido;
        stringAImprimir = stringAImprimir + " Fecha y hora: " + fechaHora;
        write(stringAImprimir,registroDeAcciones);
    }
}

void informarError(error, embebido){
    //informo el error encontrado, esto se realiza sobre un mismo log
    //utilizo region critica para que solo un hilo a la vez pueda escribir sobre
    //el archivo Log
    DateTime fechaHora = DateTime();
    #pragma omp critical (RegistrarError)
    {
        String stringAImprimir = "";
        stringAImprimir = stringAImprimir + "Error: "  + getError(error); //devuelve string correspondiente al Nº de error
        stringAImprimir = stringAImprimir + " ocurrido sobre el embebido: " + embebido;
        stringAImprimir = stringAImprimir + " Fecha y Hora: " + fechaHora;
        write(stringAImprimir,log);
    }
}
```

4 Pruebas que pueden realizarse

Como prueba principal a realizar, sería la medición de tiempos de respuesta que se obtienen del servidor aplicando el análisis de manera secuencial, es decir, de a una habitación a la vez, y luego implementar paralelismo con OpenMp. En base a los tiempos obtenidos realizar una comparación. En estas pruebas se asume que el dominio es el mismo.

Otras pruebas pueden ser como influye en el rendimiento del servidor, a medida que se incrementan la cantidad de habitaciones de las cuales recibe información. Así poder realizar un análisis sobre los tiempos de respuesta que se obtiene utilizando OpenMP para el paralelismo, para cada caso implementado.

Por último, seria aplicar una sobre carga de datos desde múltiples puntos, generando un caso de fatiga, para así determinar los límites del sistema y si este es capaz de funcionar con normalidad en aquellos casos.

5 Conclusiones

A través de este trabajo intentamos aumentar el alcance con el que cuenta nuestro sistema de iluminación inteligente LumusSystem, incrementando la cantidad de habitaciones que el sistema es capaz de administrar (actualmente una sola), utilizando para ello OpenMP, lo que nos permite implementar de forma sencilla y segura el paralelismo para el procesamiento de los distintos ambientes.

Durante la investigación pudimos aprender cómo resolver situaciones en las que un mismo código que será ejecutado de manera paralela, sea capaz de recibir información y comunicarse con distintos embebidos.

Para futuros trabajos se podría utilizar la información registrada por el servidor de las diferentes acciones en cada embebido, para generar estadísticas sobre: el consumo de energía durante un periodo de tiempo, cantidad de horas de luz natural, ingresos registrados a los espacios. Y en base a dicho análisis, el sistema pueda ajustar las acciones a realizar, como también informar al usuario a través de una interfaz, los diferentes datos extraídos.

6 Referencias

1.http://www.dalkescientific.com/writings/diary/archive/2012/01/13/openmp_vs_posix_threads.html ”OpenMp vs POSIX threads”

2. http://www.cs.colostate.edu/~cs675/OpenMPvsThreads.pdf "OpenMp vs threads”

3. https://www.openmp.org/wp-content/uploads/omp-hands-on-SC08.pdf "A Hands-on Introduction to OpenMP"

4. http://sedici.unlp.edu.ar/bitstream/handle/10915/50188/Documento_completo.pdf-PDFA.pdf?sequence=1 “Análisis del impacto de distintas técnicas de optimización de rendimiento en multicore”

5. http://www.trp.org.in/wp-content/uploads/2015/10/AJCST-Vol.4.No_.1-Jan-June-2015-pp.8-13.pdf “Review of the Parallel Programming and its Challenges on the Multicore Processors”

6. https://www.epcc.ed.ac.uk/sites/default/files/PDF/ewomp99paper.pdf “Measuring Synchronication and Scheduling Overheads in OpenMP”