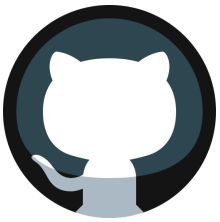
FRANKLIN AGUIRRE ORTIZ 1841743-3743

INFORME

Recursos de entrega de la practica:



https://github.com/frank1192/ExamenFranklinPractica.git

imagenes editadas

HARDWARE DE LA MÁQUINA
32 de ram
COMANDO : cat /proc/cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 151
model name : 12th Gen Intel(R) Core(TM) i5-12600K
stepping : 2
microcode : 0xffffffff
cpu MHz : 3686.399
cache size : 20480 KB
physical id : 0
siblings : 16
core id : 0
cpu cores : 8
apicid : 0
initial apicid : 0
fpu : yes
fpu_exception : yes
cpuid level : 28
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss ht syscall nx pdpe1gb rdtscp lm constant_tsc rep_good nopl xtopology tsc_reliable nonstop_tsc cpuid pni pclmulqdq vmx ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch invpcid_single ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi ept vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec xgetbv1 xsaves umip waitpkg gfni vaes vpclmulqdq rdpid movdiri movdir64b fsrm md_clear serialize flush_l1d arch_capabilities
vmx flags : vnmi invvpid ept_x_only ept_ad ept_1gb tsc_offset vtpr ept vpid unrestricted_guest ept_mode_based_exec tsc_scaling usr_wait_pause
bugs : spectre_v1 spectre_v2 spec_store_bypass swapgs
bogomips : 7372.79
clflush size : 64

cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual

Tomar los tiempos que toma procesar y filtrar el conjunto de imágenes descargadas. Tomar tiempos de la versión del programa en secuencial. Correr 5 veces el procesamiento de todas las imágenes y eliminar el tiempo más bajo y el más alto observado. Sacar el promedio y ese será el tiempo de ejecución de la versión secuencial.

PRUEBA 1
real   0m7.667s

PRUEBA 2
~~real   0m7.399s~~

PRUEBA 3
real   0m7.566s

PRUEBA 4
~~real   0m7.849s~~

PRUEBA 5
real   0m7.533s

PROMEDIO : 7.59

prueba con todos los hilos

PRUEBA 1
real 0m7.100s

PRUEBA 2
real 0m7.144s

PRUEBA 3
real 0m7.162s

PRUEBA 4
~~real 0m7.090s~~

PRUEBA 5
~~real 0m7.237s~~

Promedio : 7.14 segundos.

$$Speedup = \frac{Tiempo\ Secuencial}{Tiempo\ Paralelo}$$

Calculo speed up : 1,063

$$eficiencia = \frac{Speedup}{número\ de\ hilos}$$

eficiencia = 0,066 = 6,6%

PRUEBA 1
~~real 0m7.850s~~

PRUEBA 2
real 0m8.742s

PRUEBA 3
real 0m10.861s

PRUEBA 4

real 0m13.670s

PRUEBA 5

~~real 0m20.549s~~

Promedio = 11.091

$$Speedup = \frac{Tiempo\ Secuencial}{Tiempo\ Paralelo}$$

Cálculo speed up : 0,684

$$eficiencia = \frac{Speedup}{número\ de\ hilos}$$

eficiencia = 0,021 = 2,1%

Que podemos inferir de esto ??

CONLUSIONES

Como comentario principal, solo mirando los promedios vimos que tenemos una leve mejora de promedio de tiempos cuando usamos todos los hilos, pero no tan significativa , Los tiempos de ejecución de las versiones secuencial y paralela muestran que la paralelización no está proporcionando la ganancia esperada a nivel de una mejora sustancial.

La versión paralela con 32 hilos, el speedup es **0.684**, indicando que la versión paralela es más lenta que la secuencial en algunas pruebas el speed up ideal l sería cercano a 1 (o al 100%), lo que indicaría que cada hilo contribuye plenamente a acelerar la ejecución. Con una eficiencia de 2.1%), lo cual nos da una eficiencia muy baja ya que por cada hilo adicional, el rendimiento no mejora como debería, lo que sugiere ineficiencia en la gestión de los hilos, con lo mas probable es que hay un overhead de sincronización que significa que la gestión y coordinación entre tantos hilos puede estar creando una carga adicional.

ante estas pruebas podemos concluir también esto

la paralelización como lo hemos visto normalmente se usa para problemas muy muy grandes con una alta carga de procesos, el problema que veo, es que con el all.sh, va cogiendo imagen por imagen, y pues como entendí el problema la idea era paralelizar el main.c, por lo cual, si desde ahi parte el problema, lo ideal sería manejar una imagen por hilo y que este la modifique, pero en el archivo main.c puede que si se pudiera hacer, pero pues creo que no era la idea de la práctica

Este problema desde un inicio tiene una escalabilidad limitada: Si las tareas no son lo suficientemente grandes o el procesamiento de cada imagen no se beneficia tanto del paralelismo, se puede generar una saturación donde los hilos adicionales no ofrecen mejoras.

esto seria todo, por ahi investigue y habia una forma de paralelizar el all.sh pero estamos viendo openmp asi que nada que ver pero la dejo jeje

sudo apt-get install parallel

```C/C++
#!/usr/bin/env bash
#
# Este script se encarga de invocar los tres programas que permiten la
# conversión de múltiples archivos JPG a secuencia de píxeles, procesarlos
# y posteriormente convertir esa secuencia a archivos PNG.
#
# Autor: John Sanabria - john.sanabria@correounivalle.edu.co
# Fecha: 2024-08-22
#

procesar_imagen() {
  INPUT_JPG="$1"
  TEMP_FILE="${INPUT_JPG%.jpg}.bin"

  echo "Procesando ${INPUT_JPG}..."
  python3 fromPNG2Bin.py "${INPUT_JPG}"
  ./main "${TEMP_FILE}"
  python3 fromBin2PNG.py "${TEMP_FILE}.new"
}

export -f procesar_imagen
# Usamos GNU parallel para procesar imágenes en paralelo
find imagenes/*.jpg | parallel procesar_imagen
```

con eso mejoraba el programa hasta 1 o 3 segundos, pero reitero esa no es la finalidad de la practica, pero es un dato que me gustaria dejar

link al repositorio por si acaso nuevamente
https://github.com/frank1192/ExamenFranklinPractica.git