Arquitectura de Computadores

2º Curso Grao Enx. Informática

Práctica 2

Programación Multinúcleo e extensións SIMD

Obxectivos: programar un algoritmo simple con vectores representados en punto flotante, utilizando diferentes graos de optimización (utilización de varios núcleos, utilización de extensións vectoriais simd, estratexias para reducir os fallos cache,..) e tomar medidas de rendemento combinando diferentes optimizacións e tamaños do problema.

Equipos: grupos de prácticas de ata dúas persoas.

Tempo de traballo presencial: 5 sesións de prácticas.

Prazo de entrega: Lúns 13 de Maio 2019 ata as 14:00.

Valoración: 60% da nota de prácticas. Valoración por apartados (sobre 10): apartado i) e ii) → 1 punto, apartado iii) → 4.5 puntos facendo as opcións a) e b), 3 puntos facendo só unha das opcións, apartado iv) → 4.5 puntos.

Facer diferentes programas en C que realicen a computación do quaternion dp a partir dos vectores de quaternions a e b, indicado polo seguinte pseudocódigo:

Entradas:

```
vectores de quaternions a(N),b(N): N elementos de valor aleatorio coas
compoñentes dos quaternions de tipo float.
```

Saída:

```
dp: quaternions con componentes de tipo float.
```

Computación:

```
c(N): vector auxiliar de quaternions.
for i=1,N {
    c(i)=a(i)*b(i); // * indica multiplicación de quaternions.
}
dp=0; //inicialización do quaternion a cero (todas as compoñentes cero);
for i=1,N {
    dp=dp+c(i)*c(i) // * e +: multiplicación e suma sobre quaternions.
}
```

As diferentes versións son as seguintes:

i) Programa secuencial base (codificación en C do pseudocódigo anterior utilizando para cada quaternion catro variables tipo float)

ii) Programa secuencial optimizado (intentar modificar o código – o único que nos interesa é o valor final de dp - para obter algunha mellora no tempo de execución se é posible – cache, paralelismo a nivel de instrucción, simplificación de operacións, etc)

iii) Programa secuencial optimizado utilizando extensións SSE para utilizar procesamento vectorial simd, seguindo a mellor (implementación con menor retardo) das seguintes estratexias: iii-a) vectorizar a multiplicación de dous quaternions, iii- b) vectorizar por iteracións do bucle

iv) Programa utilizando OpenMP para paralelizar a version secuencial optimizada (**non está permitido utilizar variables tipo "reduction"**), sen utilizar as extensións SSE, e variando o número de fíos (cores): 1, 2, 4, 8 e 16.

Os códigos deberán compilarse do seguinte xeito:

i) Este código deberá compilarse sen optimizacións do compilador: "`icc -O0 file_name.c`", e con optimizacións incluída a autovectorización "`icc -O2 file_name.c`"
ii) Este código deberá compilarse sen optimizacións do compilador: "`icc -O0 file_name.c`"
iii) Estes códigos deberán compilarse sen optimizacións do compilador: "`icc -O0 file_name.c`"
iv) Estes códigos deberán compilarse sen optimizacións do compilador e co flag de OpenMP: "`icc -O0 -qopenmp file_name.c`"

Facer varios experimentos utilizando os seguintes tamaños de vectores: $N=10^q$, con q=2, 4, 6 e 7. En todos os casos inicializar os vectores con valores aleatorios, e medir o número de ciclos da parte do programa na que se fai a computación indicada anteriormente. O almacenamento inicial dos datos debe ser igual en todas as versións. Cando se utilice OpenMP ou extensións SSE, debe incluírse na medida de tempo todo o que implique unha sobrecarga respecto do programa secuencial optimizado. Para cada caso, tomar 10 medidas, e seleccionar a mediana destes valores como valor final da medida.

As medidas de ciclos deben tomarse no supercomputador Finisterrae2 do CESGA utilizando un nodo con 2 procesadores Intel Xeon de arquitectura Haswell (as medidas deben realizarse sempre no mesmo tipo de nodos).

Solicitar ao profesor unha conta para acceder ao CESGA. Utilizar as rutinas de medida de ciclos da Práctica 1.

MEMORIA DA PRÁCTICA: debe entregarse unha memoria (en papel) cos resultados da práctica. A memoria debe conter unha introdución, que describa os obxectivos dos experimentos e as características do procesador utilizado. Para cada apartado debe conter o listado do código (non incluír as rutinas de medidas de ciclos), as gráficas obtidas, e unha interpretación das mesmas. Finalmente deben presentarse as conclusións finais a modo de resumo. ***Facer as representacións gráficas que se consideren oportunas para sacar conclusións dos resultados obtidos (é de especial interese a ganancia en velocidade da versión secuencial optimizada con respecto á versión inicial, a ganancia en velocidade dos códigos dos apartados iii), iv) e v) con respecto da versión secuencial optimizada, e finalmente a versión inicial compilada con -O2 con respecto a todas as demais).***

CÓDIGO FONTE: os códigos fonte resultado dos apartados ii), iii) e iv) envíaranse o profesor dentro dun arquivo zip ou rar a través do Campus Virtual. **O envío é obrigatorio** (e dentro do prazo de entrega da memoria). **En caso de facer a práctica en grupo SÓ un membro do grupo debe enviar o código.**

CRITERIOS DE AVALIACIÓN: cumprimento das especificacións do enunciado da práctica, buscando sempre obter a mínima latencia do programa nas diferentes implementacións, rigurosidade no plantexamento e interpretación de resultados, exploración de alternativas, estudo autónomo e presentación de resultados (calidade da memoria). **Será considerado como moi negativo a copia directa de código (dende calquera fonte).**

INSTRUCCIÓNS PARA TRABALLAR NO CLÚSTER FT2 DO CESGA (en inglés):

Pódese consultar esta e outra información en:

https://www.cesga.es/gl/infraestructuras/computacion/FinisTerrae2

Machines to run the experiments: node containing two Haswell processors of the

FT2 cluster at CESGA (ft2.cesga.es).

Compiler: Intel icc

Connection:

Make the ssh connections to FT2 using:

ssh cursoXXX@ft2.cesga.es

File transfers: you can use a linux client to transfer files between CESGA and your computer. An alternative is to use scp:

scp local_file_name cursoXXX@ft2.cesga.es:destination_file_name

Compiling and running programs in the FT2 cluster:

1-Connect to FT2 cluster using ssh.

2-You should have transferred the file with code to compile.

3-Load the necessary packets for compilation: "module load intel". If you want to know the available packets type "module av".

5-Compile your program following the instructions detailed above.

6-Make a script (for instance "my_work.sh") with the following contents:

```
#!/bin/bash

#SBATCH -n 1
#SBATCH -c 20
#SBATCH -p shared
#SBATCH --qos=shared_short
#SBATCH -C has2s
#SBATCH -t 00:00:30

module load intel
srun ./executable_file
```

Save the script and allow permissions of execution to the file.

```
#SBATCH -t 00:00:30 specifies the maximum execution time in HH:MM:SS
#SBATCH -c 20 indicates the maximum number of cores (threads) to use
```

7-Submit the script to the queue with the command "qbatch":

"sbatch my_work.sh"

This would run the contents of the script my_work.sh for a maximum of 30 seconds with a maximum of 20 cores.

8-After some time (you can check if your work is still in the queue with "squeue -u cursoXXX"), you will get one file in your account: "slurm-ID_OF_THE_WORK.out" with the actual output of the program and some additional information.