PD08

Thursday, November 17, 2022 6:02 PM

- Utilice el código de la integral por el método de cuadratura adjunto. Compile y ejecute el código en MPI y OMP. Mida tiempos de ejecución. Si es necesario, modifique la carga del algoritmo para medir tiempos significativos

Codigos adjuntos con scripts en python para las pruebas en khipu.

- realice las pruebas en el cluster Khipu con hasta 16 hilos OMP y hasta 12 procesos MPI (si puede usar más recursos el resultado será mejor)

Resultados de pruebas en khipu:

```
SERIAL, n: 1000000
   processors    |         average(s)
-------------------------------------
         1       |   0.03150000000000001410
SERIAL, n: 10000000
   processors    |         average(s)
-------------------------------------
         1       |   0.29000000000000003553
SERIAL, n: 100000000
   processors    |         average(s)
-------------------------------------
         1       |   2.88500000000000039790
```

```
MPI, n: 1000000
   processors    |        average(s)
-------------------------------------
         1       |   0.00749999999999999972
         2       |   0.00950000000000000150
         4       |   0.00649999999999999883
         8       |   0.00400000000000000008
         12      |   0.00150000000000000003
MPI, n: 10000000
   processors    |        average(s)
-------------------------------------
         1       |   0.09000000000000002442
         2       |   0.04800000000000000794
         4       |   0.03000000000000000930
         8       |   0.02000000000000000389
         12      |   0.01750000000000000167
MPI, n: 100000000
   processors    |        average(s)
-------------------------------------
         1       |   0.90950000000000008615
         2       |   0.46350000000000007860
         4       |   0.23800000000000007261
         8       |   0.12650000000000000133
         12      |   0.08500000000000001998
```

```
OMP, n: 1000000
   processors    |        average(s)
-------------------------------------
         1       |   0.03315734999999999516
         2       |   0.01540054999999999549
         4       |   0.01163144999999999978
         8       |   0.00476340000000000010
         12      |   0.00474320000000000037
         14      |   0.00514510000000000054
         16      |   0.00400109999999999962
OMP, n: 10000000
   processors    |        average(s)
-------------------------------------
         1       |   0.29435185000000002598
         2       |   0.14718309999999998339
         4       |   0.07492914999999998604
         8       |   0.04038895000000000679
         12      |   0.02648364999999999747
         14      |   0.02292825000000000071
         16      |   0.01994814999999999783
OMP, n: 100000000
   processors    |        average(s)
-------------------------------------
         1       |   2.88834670000000048873
         2       |   1.44439194999999953595
         4       |   0.72564375000000003180
         8       |   0.36447324999999997086
         12      |   0.24642179999999996864
         14      |   0.21072004999999999209
         16      |   0.18314869999999999761
```

- Calcule la velocidad de ejecución en paralelo (en FLOPs) para cada caso, e.g. contabilizando las operaciones de coma flotante y dividiéndolas entre el tiempo

```
// quad_serial
total = 0.0;
  for (i = 0; i < n; i++)
  {
    x = ((n - i - 1) * a + (i)*b) / (n - 1);
    total = total + f(x);
  }

double f(double x)
{
  double pi = 3.141592653589793;
  double value;
  value = 50.0 / (pi * (2500.0 * x * x + 1.0));
  return value;
}
```

Operaciones dentro del for:

6 + 1 + 1 + 1 + 1 = 10

Reduccion (MPI y OMP):

p

Num operaciones:

10n + p

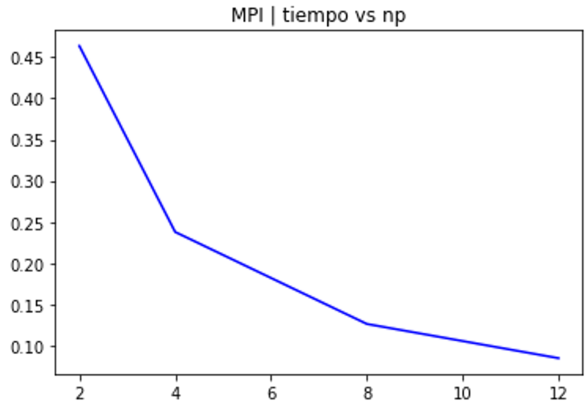
usando n = 10^8

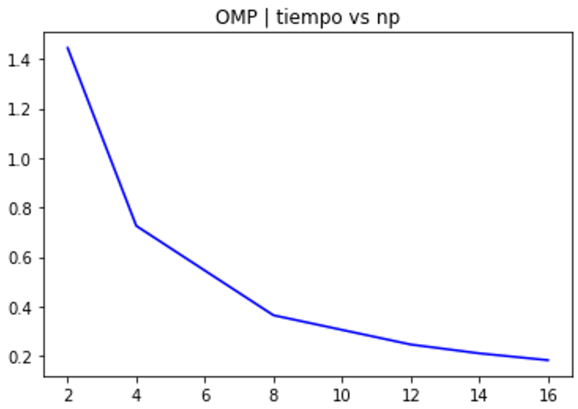
FLOPS:

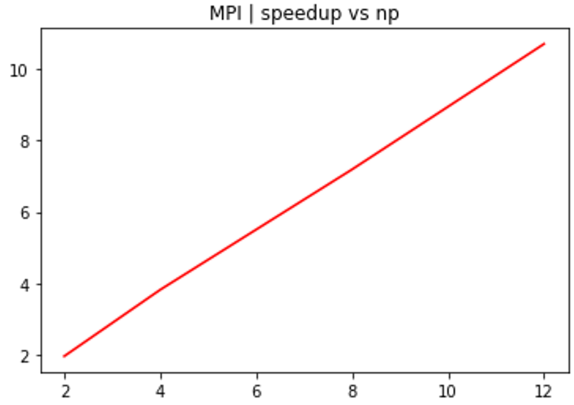
```
MPI
(10*100000000 + 2) / 0.46350000000000007860 = 2157497307.443365/s
(10*100000000 + 4) / 0.23800000000000007261 = 4201680689.0756288/s
(10*100000000 + 8) / 0.12650000000000000133 = 7905138403.162055/s
(10*100000000 + 12) / 0.08500000000000001998 = 11764706023.52941/s

OMP
(10*100000000 + 2) / 1.44439194999999953595 = 692332854.6659377/s
(10*100000000 + 4) / 0.72564375000000003180 = 1378086704.39179/s
(10*100000000 + 8) / 0.36447324999999997086 = 2743685601.069489/s
(10*100000000 + 12) / 0.24642179999999996864 = 4058082572.2399564/s
(10*100000000 + 14) / 0.21072004999999999209 = 4745632957.091648/s
(10*100000000 + 16) / 0.18314869999999999761 = 5460044302.798764/s
```

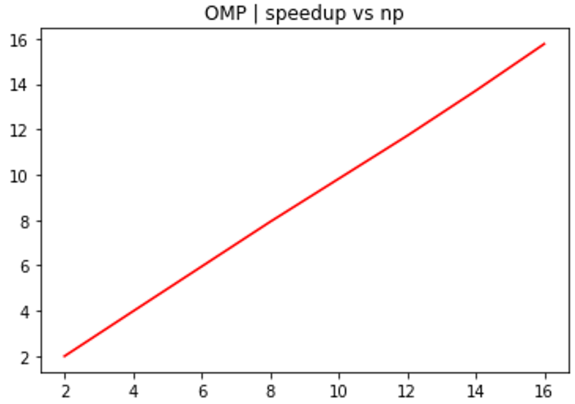
- presente los resultados en sendos graficos **t** vs **np**, **s** vs **np**.

usando n = 10^8









- Determine la eficiencia del algoritmo al dividir la velocidad entre np (número de procesos). Realice este calculo tanto con MPI como OMP
- realice pruebas de escalabilidad fuerte y débil. Utilice el principio de isoeficiencia para determinar la relación entre n y p. Comente si se puede observar esta relación en los resultados experimentales obtenidos.
- Concluya y comente sobre la escalabilidad del algoritmo

OMP:

Tp = O(n/p + p)

S = n/(n/p + p)

E = (n/(n/p + p))/p

$$\mathrm{E} = \mathrm{O}\left(\frac{1}{1+\frac{p^2}{n}}\right), E(1) \rightarrow n = p^2$$

MPI (incluyendo comunicacion):

Tp = O(n/p + p + log(p))

$$\mathrm{E} = \mathrm{O}\left(\frac{1}{\quad}\right), E(1) \rightarrow n = p^2$$

MPI (incluyendo comunicacion):

Tp = O(n/p + p + log(p))

S = n/(n/p + p + log(p))

E = (n/(n/p + p + log(p)))/p

$$\mathrm{E} = \mathrm{O}\left(\frac{1}{1+\frac{p^2}{n}+\frac{\mathrm{plog}\,p}{n}}\right), E(1) \rightarrow n = p^2$$

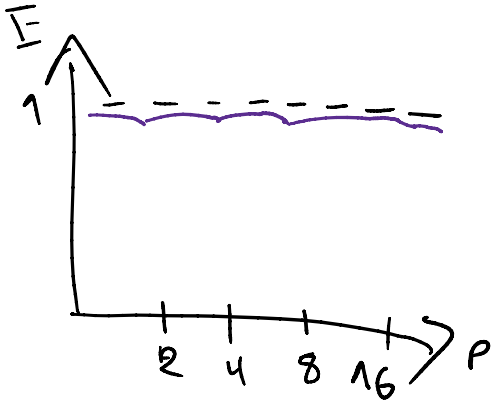
En base a los resultados observados en la experimentación, y los resultados de speedup y eficiencia, podemos determinar que OMP y MPI tienen un comportamiento muy cercano a una mejora lineal con respecto a p, resultado que podríamos apreciar en las gráficas de tiempo vs np.

El speedup experimental que se consigue es casi lineal en ambos casos, al igual que el speedup teórico que se acerca igualmente a un caso lineal.

La condicion de isoeficiencia encontrada para OMP y MPI es ambos de p^2, incluso al considerar el overhead de comunicación que presenta MPI.

Con respecto a la escalabilidad débil, vemos en las dos gráficas de speedup anteriores para n = 10^8, el crecimiento experimental resulta ser casi lineal en ambos casos, conforme crece el numero de procesos.

Con respecto a la escalabilidad fuerte, vemos como el algoritmo cuando incrementa o decrece el n, obteniendo un decenso de running time casi idéntico para n = 10^6, 10^7 y 10^8. Una gráfica de eficiencia en este caso seguiría un comportamiento casi constante:



Podemos decir entonces que la escalabilidad débil y fuerte tienen un comportamiento muy favorable para las implementaciones paralelas de este algoritmo.