

Computación Paralela y Distribuida

2022-II

José Fiestas

22/08/22

Universidad de Ingeniería y Tecnología
jfiestas@utec.edu.pe

Objetivos:

1. Velocidad, eficiencia, escalabilidad. Ley de Ahmdal
2. Modelos computacionales en paralelo (PRAM)

Al finalizar la unidad, los alumnos conocen:

1. Implicaciones de la ley de Amdahl para un algoritmo paralelo
2. Definir velocidad de un algoritmo paralelo, y explicar la noción de escalabilidad

Importancia del paralelismo y estrategias.

- paralelismo debe enfocarse a todo nivel: algoritmo, código, sistema operativo, compilador, hardware
- Se debe considerar el tiempo de comunicación entre procesos así como de proceso a memoria. Se diferencian entonces algoritmos limitados por **tiempo de ejecución** por proceso, de los limitados por **comunicación** entre procesos
- por razones prácticas, el software se adapta al hardware

**Velocidad, eficiencia,
escalabilidad. Ley de Ahmdal**

Velocidad, eficiencia, escalabilidad

El tiempo de ejecución secuencial (normalizado a la unidad) es:

$T_s = f_s + f_p$, con la fracción de código secuencial f_s , y paralela f_p
($f_s + f_p = 1$)

Resolverlo en p procesadores, implica que:

$$T_p = f_s + \frac{f_p}{p}$$

strong scaling

Si definimos **performance** como **trabajo por tiempo**:

$$P_s = \frac{f_p + f_s}{T_s} = 1$$

mientras que en paralelo

$$P_p = \frac{f_p + f_s}{T_p} = \frac{f_p + f_s}{f_s + (1 - f_s)/p} = \frac{1}{f_s + (1 - f_s)/p}$$

Y midiendo la velocidad, como:

$$S = \frac{P_p}{P_s} = \frac{1}{f_s + f_p/p}$$

weak scaling

Si definimos el tiempo de ejecución para un problema variable, tenemos:

$$T_s = f_s + f_p p^\alpha$$

$$T_p = f_s + f_p p^{\alpha-1}$$

Siendo el performance:

$$P_s = \frac{f_s + f_p p^\alpha}{T_s} = 1$$

y en paralelo:

$$P_p = \frac{f_s + f_p p^\alpha}{T_p} = \frac{f_s + f_p p^\alpha}{f_s + f_p p^{\alpha-1}}$$

Siendo la velocidad :

$$S = \frac{P_p}{P_s} = \frac{f_s + f_p p^\alpha}{f_s + f_p p^{\alpha-1}} = P_p$$

Para $\alpha = 0$ obtenemos **strong scaling**

Eficiencia:

Se trata del poder computacional en paralelo

$$\epsilon = \frac{\text{performance } p \text{ procesos}}{p \times \text{performance en 1 proceso}} = \frac{\text{speedup}}{p}$$

Considerando weak scaling, ya que en el límite $\alpha \rightarrow 0$ deriva en strong scaling. Si el trabajo es $f_s + f_p p^\alpha$, obtenemos:

$$\epsilon = \frac{S}{p} = \frac{f_s + (1-f_s)p^\alpha}{(f_s + (1-f_s)p^{\alpha-1}) \times p} = \frac{f_s p^{-\alpha} + (1-f_s)}{f_s p^{1-\alpha} + (1-f_s)}$$

$$\text{Para } \alpha = 0, \epsilon = \frac{1}{f_s p + (1-f_s)}$$

$$\text{Para } \alpha = 1, \epsilon = f_s p^{-1} + (1-f_s)$$

En **strong scaling**, se deriva ...

Ley de Amdahl

Intenta responder la pregunta sobre que tan rápido ejecuta un código en paralelo cuando utilizo p procesos. El speedup se incrementa con p .

$$S = \frac{1}{f_s + \frac{1-f_s}{p}}$$

Y la eficiencia $E = \frac{1}{p(f_s + \frac{1-f_s}{p})}$.

Para $p \rightarrow \infty$, $S = 1/f_s$ (limitado por la parte secuencial)

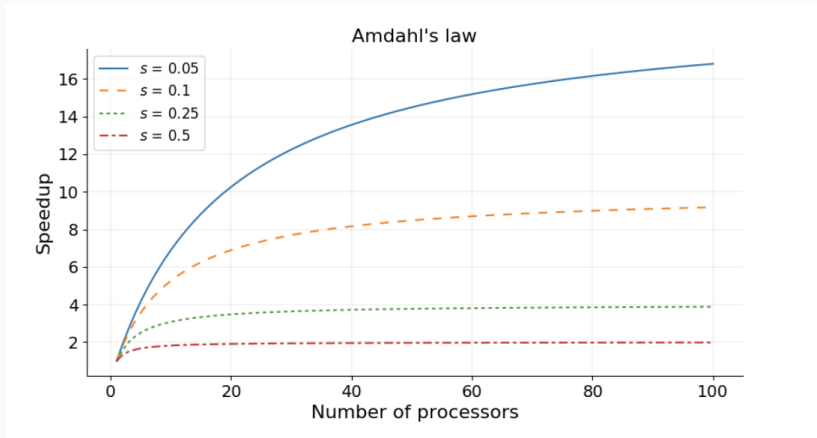


Figura 1: *

Scalability: strong and weak scaling. Xin Li, 2018

En **weak scaling**, se deriva . . .

Ley de Gustafson

Intenta responder la pregunta sobre que tan rápido ejecuta un código en paralelo cuando ejecuto un problema de tamaño variable (creciente) en p procesos.

$$S = \frac{P_p}{P_s} = \frac{f_s + f_p p^\alpha}{f_s + f_p p^{\alpha-1}}$$

La fracción en paralelo escala con el número de procesos p .

$$\text{Para } \alpha = 0, S = \frac{f_s + f_p}{f_s + f_p/p} = \frac{1}{f_s + f_p/p}, \text{ (Ley de Amdahl)}$$

$$\text{Para } \alpha = 1, S = \frac{f_s + f_p p}{f_s + f_p} = f_s + f_p p$$

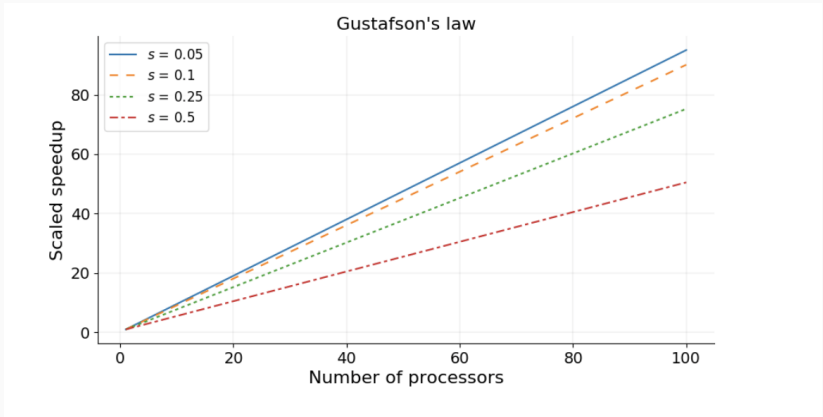


Figura 2: *

Scalability: strong and weak scaling. Xin Li, 2018

La eficiencia es la **carga promedio** por procesador:

$$E(n) = \frac{T_s}{p \cdot T_p}$$

Es decir, en general, $S \leq p$, lo que corresponde a $E \leq 1$.

Si $S=p$, $E=1$.

La escalabilidad evalúa el performance proporcional a un creciente número de procesos.

La escalabilidad mantiene una eficiencia constante si se incrementa el número de procesos (p), a la vez que la carga (n), es decir, si problemas grandes pueden ser resueltos con la misma eficiencia que problemas pequeños.

Esta propiedad no puede ser descrita por la Ley de Ahmdal, ya que esta última es independiente del número de procesos.

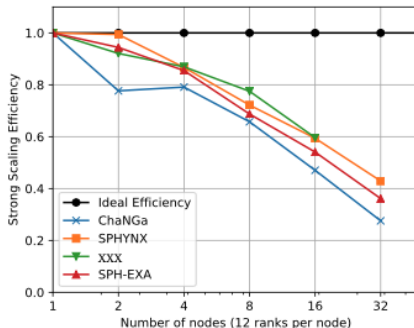


Figure 2: Parallel efficiency of ChaNGa(12 OpenMP tasks and 1 MPI rank per node), SPHYNX (12 OpenMP tasks and 1 MPI rank per node), XXX (12 MPI ranks per node) and SPH-EXA mini-app (12 MPI ranks per node) under strong scaling, in function of the number of nodes used (12 cores per node) on Piz Daint.

Figura 3: *Speedup de modelos SPH en 32 nodos*

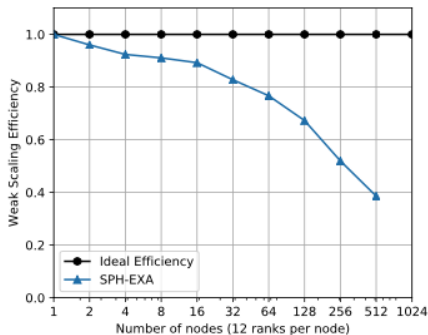


Figure 3: Parallel efficiency of the SPH-EXA mini-app (12 MPI ranks per node) under weak scaling, in function of the number of nodes used (12 cores per node) on Piz Daint.

Figura 4: *Speedup de modelos SPH en 1024 nodos*

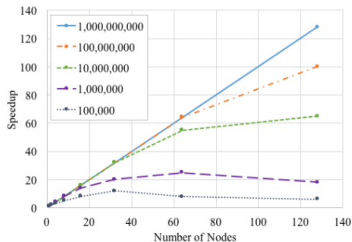


Figure 11: Strong scaling of the distributed N-Body application implemented with HPX on up to 128 nodes (20 cores per node). It shows a close to perfect HPX scalability for problem size of 10^9 particles.

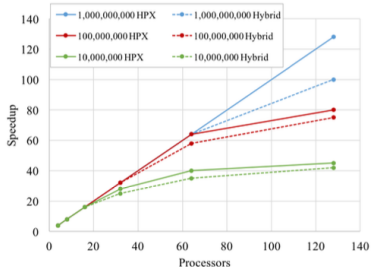






Figure 13: Comparison results of the strong scaling between HPX and a hybrid model (OpenMP/MPI), with up to 128 nodes (20 cores per node). The results illustrate a better performance for the HPX application for larger number the nodes, which is due to using a *future*-based request buffer between the remote nodes.

Figura 5: *Speedup de modelos NBody en 120 nodos*

-  David B. Kirk and Wen-mei W. Hwu *Programming Massively Parallel Processors: A Hands-on Approach*. 2nd. Morgan Kaufmann, 2013. isbn: 978-0-12-415992-1.
-  Norm Matloff. *Programming on Parallel Machines*. University of California, Davis, 2014.
-  Peter S. Pacheco. *An Introduction to Parallel Programming*. 1st. Morgan Kaufmann, 2011. isbn: 978-0-12-374260- 5. *** Ch 2.6
-  Michael J. Quinn. *Parallel Programming in C with MPI and OpenMP*. 1st. McGraw-Hill Education Group, 2003. isbn: 0071232656. *** Ch 7.1-5

*** en esta clase