UNCUYO
UNIVERSIDAD NACIONAL DE CUYO
FACULTAD DE INGENIERÍA

Licenciatura en Ciencias de la Computación

Arquitectura de Computadoras II

Unidad 5

Modelos y Arquitecturas Escalables

Dr. Ing. Pablo Daniel Godoy (pablo.godoy@ingenieria.uncuyo.edu.ar)

Multicomputadoras

- Problemas de los sistemas multinúcleo:
  - Problemas de **coherencia memoria caché difícil de resolver** (debido a que hay una memoria principal, RAM, compartida).
  - **Difícil de escalar**, escalabilidad limitada (no se puede agregar núcleos a un chip ya construido).
- **Multi-computadora**: Sistema formado por varias **computadoras completas interconectadas** (a traves de una red) + **middleware** que permite que resuelvan en conjunto un problema.

Sistemas multi-computadora

- Paralelismo a nivel de computadoras.
  - Sistemas multicomputadoras **fuertemente acoplados** (cluster)
    - Objetivo: **speedup**
    - Sistema preparado para **mucha comunicación** entre los procesos.
    - Constructivamente:
      - Las computadoras están geográficamente cerca (en una misma sala).
      - Redes de interconexión específicas de alta velocidad.
      - Adaptaciones en las arquitecturas de la computadoras.
  - Sistemas multicomputadoras **débilmente acoplados** (sistemas distribuidos, cloud computing, grid computing).
    - Objetivo: **compartir recursos**.
    - Poca comunicación entre los procesos.
    - Constructivamente: distribuidas geográficamente (diferentes partes del planeta).

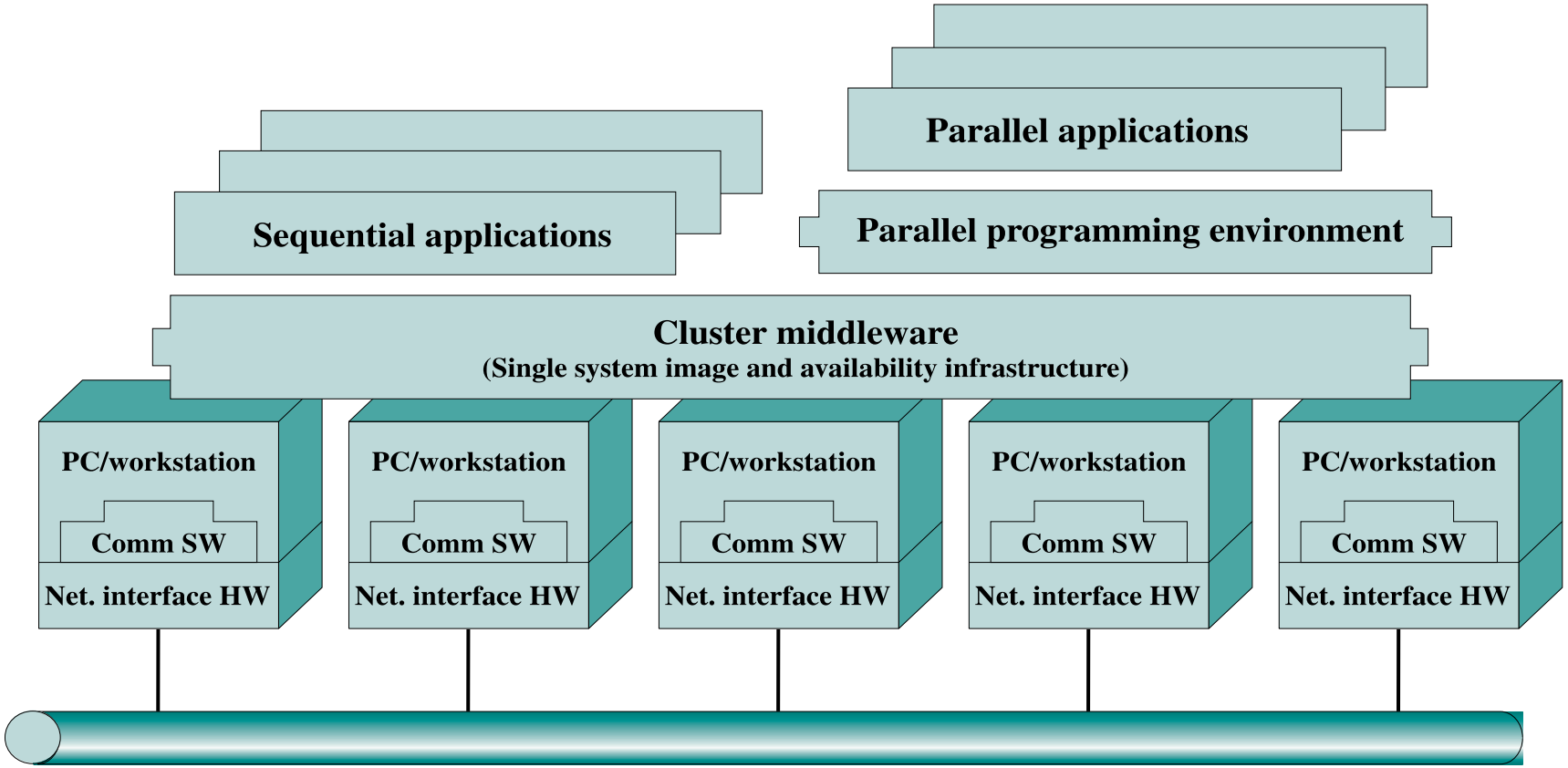


Figura basada en Computer Organization and Architecture 9th Edition William Stallings

Sistemas distribuidos

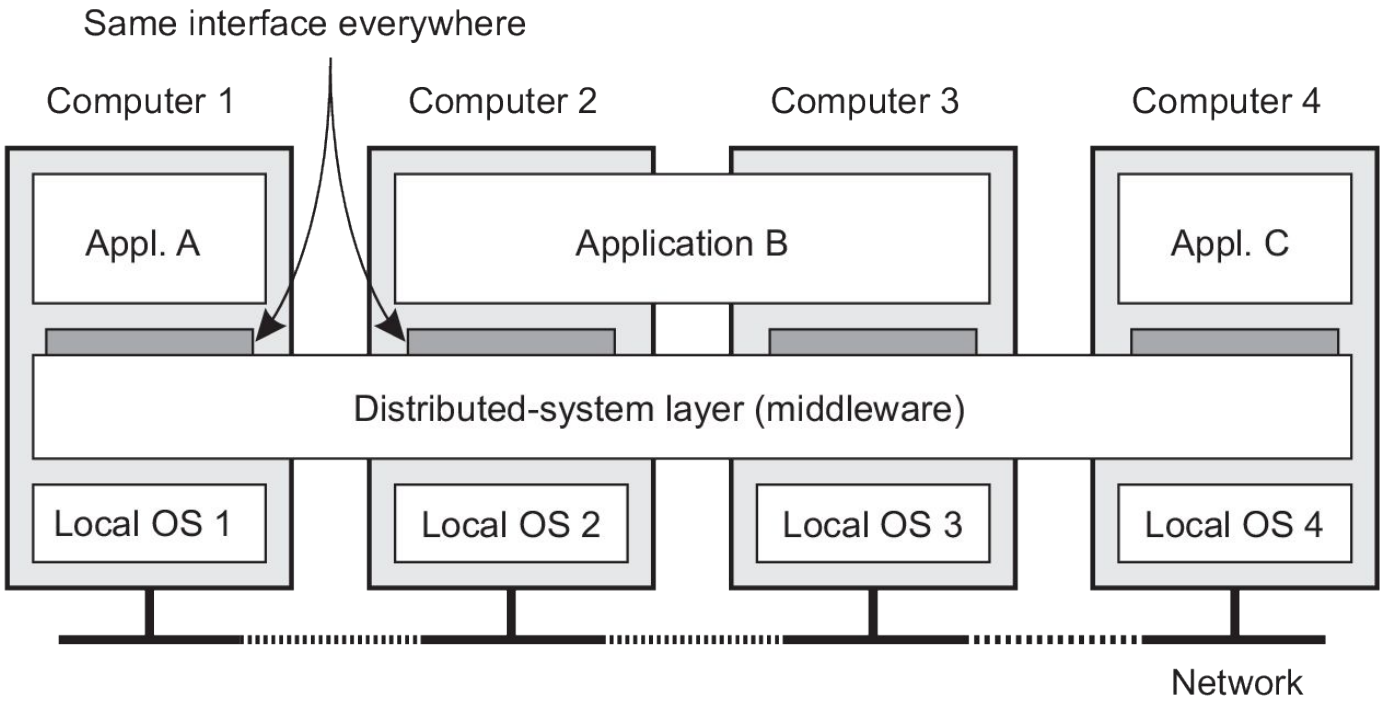


Figura obtenida de van Steen and Tanenbaum, "Distributed Systems", Third edition, v 3.03 (2020), pag 5.

Ejemplo: El Capitan

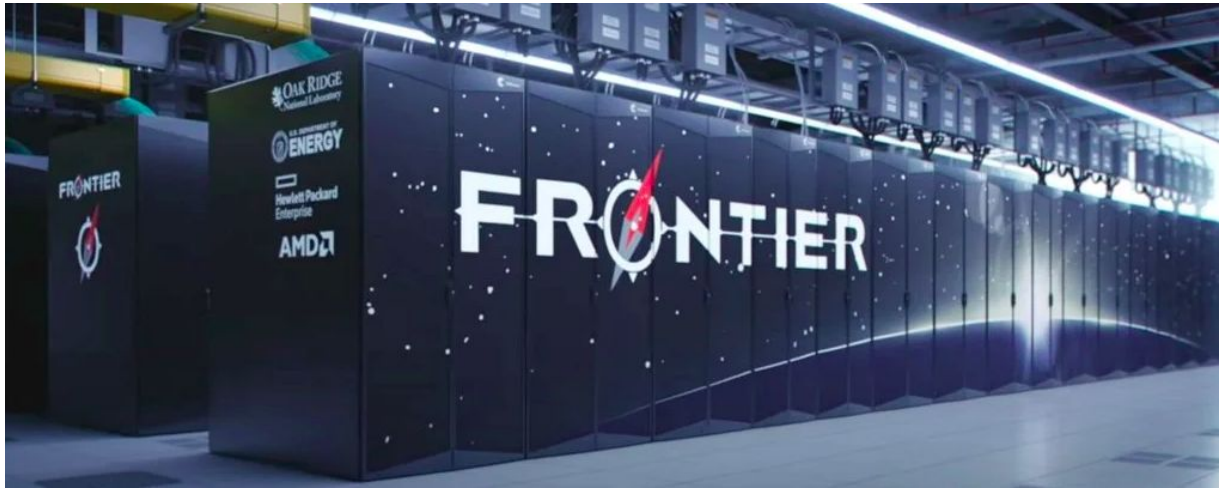
- 1° cluster más potente del mundo.
- 1.8 HexaFLOPS (pico de 2.8 HFLOPS).
- 11.3 millones núcleos AMD EPYC 1.8 GHz.
- 29.7 MW.

- HPE Cray MPI, g++.

Lawrence Livermore National Laboratory.
National Nuclear Security Administration.

https://top500.org/

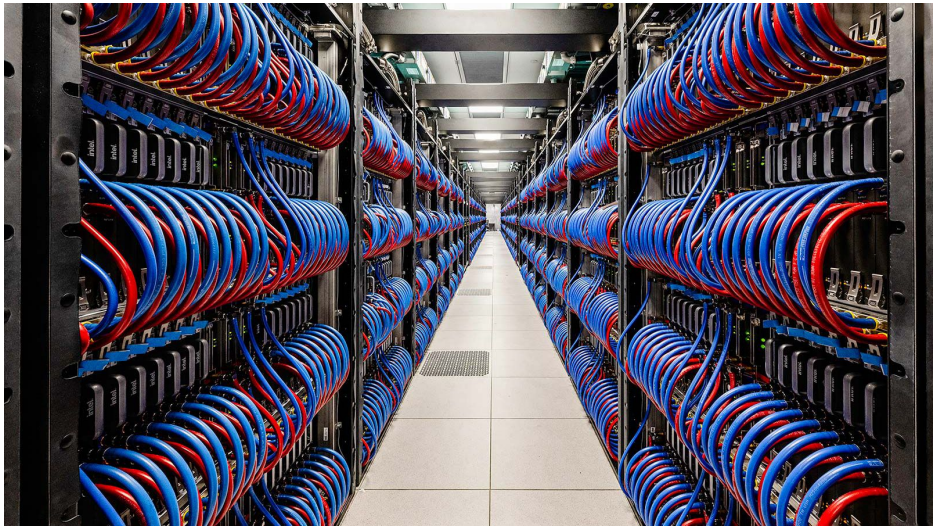
Ejemplo: Frontier



- 2° cluster más potente del mundo.
- 1.23 HexaFLOPS.
- 22.7 Mw
- 9.07 millones de núcleos AMD 3rd Generation EPYC 2GHz

- Red de interconexión: HPE Slingshot-11
- Cray-mpich, gcc/12.2.0.
- Oak Ridge National Laboratory, Estados Unidos.

Ejemplo: Aurora



- 3° cluster más potente del mundo.
- 1.01 HexaFLOPS.
- 38.6 Mw
- 9.26 millones de núcleos Xeon CPU 2.4GHz

- Red Slingshot-11
- Aurora-mpich, Intel oneAPI DPC++/C++
- Argonne Leadership Computing Facility, USA.

Ejemplo: JUPITER

- 4° cluster más potente del mundo.
- 1.0 HexaFLOPS.
- 4.8 millones de núcleos GH Superchip 72C 3GHz (Nvidia).
- Energía: 15 MW.
- NVIDIA HPC-X MPI.
- Propósito: Científico e industria.

Ejemplo: Fugaku

- 7° cluster más potente del mundo.
- 442 PetaFLOPS.
- 29 Mw
- 7.6 millones de núcleos A64FX 48C 2.2GHz. ARMv8.2-A (primer ARM de 64 bits)
- Propósito: Científico

Japón

Ejemplo: Clementina XXI

- 264° cluster más potente del mundo.
- 5.39 Peta FLOPS.

- ? Mw
- 43 mil núcleos Xeon Max 9462 32C 2.7GHz
- Argentina, Buenos Aires (SMN)
- Propósito: Científico.

Ejemplo: Serafin

- Anterior cluster más potente de argentina
- 147 Teraflops
- Centro de Cómputo de Alto Desempeño (CCAD), Córdoba
- 60 nodos.
- 3840 núcleos (120 procesadores AMD EPYC 7532).

Fuente: https://ccad.unc.edu.ar/equipamiento/cluster-serafin/

UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

FACULTAD
DE INGENIERÍA

Ejemplo: TOKO

- Cluster más potente del oeste argentino
- ITIC y FCEN, UNCuyo
- 390 núcleos (AMD Opteron 6272 de 2.1 GHz, 4 AMD Opteron 6376 de 2.3GHz, 2 AMD Epyc 7281, AMD Ryzen 7 2700/3700x)
- http://toko.uncu.edu.ar/

MPI (Message Passing Interface)

- **Especificación** para **funciones (primitivas)** y **mecanismos** para comunicación entre procesos.
  - Las funciones se implementan como **librerías** + middleware.
- Consorcio **MPI forum**: (https://www.mpi-forum.org/)
- Características:
  - Independencia del lenguaje.
  - Portabilidad del código.
  - La red subyacente no está definida en el estándar (frecuentemente es TCP/IP).
- ¿Qué hace?: **Comunicación entre procesos** a través de **mensajes** que van desde el espacio de direcciones de un proceso al espacio de direcciones del otro proceso.

Implementaciones de MPI

- Implementaciones:
  - **Open MPI**: Implementado por varias universidades de USA y Alemania. Gratuito. https://www.open-mpi.org/.
  - **MPICH**: Implementado por Mississippi State University (Argonne National Laboratory). Gratuito. https://www.mpich.org/. Aurora-mpich (versión personalizada) usara por Aurora (top 500).
  - **HPE Cray MPI**: creada por Cray Inc., luego comprada por Hewlett Packard Enterprise. Implementación basada en MPICH (utilizado por EL Capitan, Frontier y LUMI del top 500).
  - **Spectrum MPI**: Implementación paga de IBM (Utilizada por Summit del Top 500 y varias otras). https://www.ibm.com/ar-es/marketplace/spectrum-mpi
  - **Implementaciones a medida**: Fujitsu MPI (usado por la el cluster Fugaku, creda por Fujitsu). Based on OpenMPI.

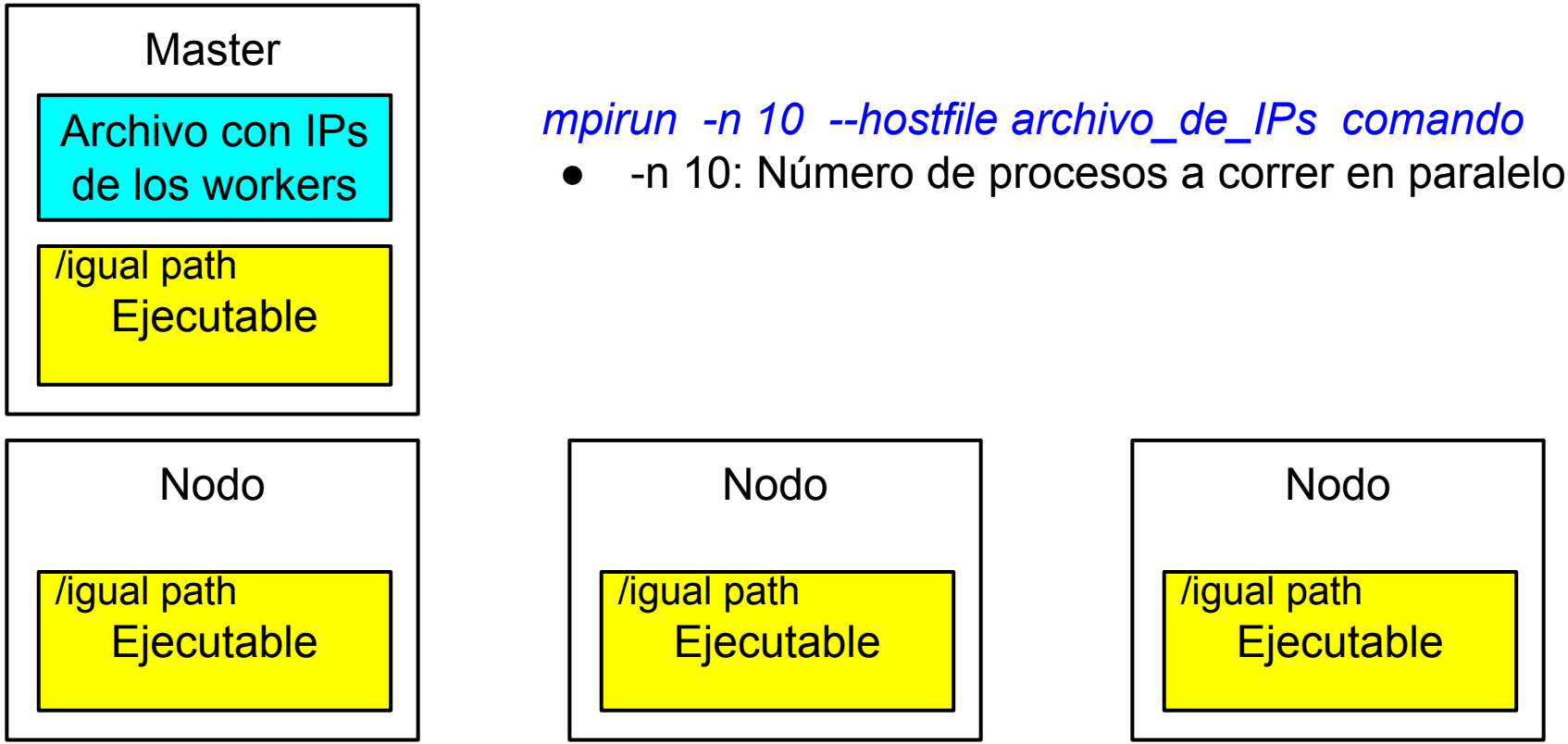
Implementación OpenMPI y MPICH

- **Middleware** que trabaja sobre **SSH**.
  - Es necesario tener ssh trabajando **sin necesidad de ingresar usuario y contraseña** en todas las máquinas del cluster:
    - El master y los nodos deben pre-compartir sus claves públicas para poder comunicarse.
- Trabajar sobre **C**, **C++** o **Fortran**.
  - Se debe tener instalado un compilador de esos lenguajes (g++ para Linux).
- Existen librerías o wrappers para otros lenguajes:
  - Por ejemplo: **MPI4PY** para Python.

Implementación OpenMPI y MPICH

- Ejecución de un programa paralelo:
  - Configurar archivo que contenga las IP de las máquinas que forman parte del Cluster en el nodo master.
  - Copiar los ejecutables en todos los nodos en el mismo directorio.
  - Ejecutar los programas paralelizables que hayamos creado en el nodo master: *mpirun -n 10 --hostfile archivo_de_ips comando*
    - -n 10: Número de procesos a correr en paralelo.
    - -f archivo_de_ips: archivos con las IPs de las máquinas que forman parte del clúster.

Ejecución de un programa sobre Open MPI y MPICH



mpirun -n 10 --hostfile archivo_de_IPs comando

- -n 10: Número de procesos a correr en paralelo.

Variables de un programa MPI

- **Comunicador:** Conjunto de procesos que resuelven una tarea comunicándose mediante MPI.
- **rank**: entero positivo que identifica a cada proceso.
- **size**: número total de procesos.

Estructura básica de un programa en C++ y Python

```cpp
#include<stdio.h>
#include <mpi.h>
using namespace std;
int main()
{
  int rank, size, length;
  char name[80];
  if(MPI_Init(NULL, NULL)!=MPI_SUCCESS){
    cout<<"Error iniciando MPI"<<endl;
    exit(1);
  }
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  MPI_Get_processor_name(name,&length);

 /*Código del programa*/

  if(MPI_Finalize()!=MPI_SUCCESS){
    cout<<"Error finalizando MPI"<<endl;
    exit(1);
  }
}
```

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()
name = MPI.Get_processor_name()

#Código del programa
```

Primitivas Send y Recv en C++

- **int MPI_Send(void *buffer, int count, MPI_Datatype datatype, int rank_dest, int tag, MPI_Comm comm)**
- **int MPI_Recv(void* buffer, int count, MPI_Datatype datatype, int rank_source, int tag, MPI_Comm comm, MPI_Status *status);**
  - buffer: buffer con datos a enviar (MPI_Send) o recibir (MPI_Recv).
  - Count: Número de datos a enviar.
  - MPI_Datatype (MPI_INT, MPI_BYTE).
  - rank_dest: proceso destino.
  - tag: etiqueta que permite distinguir distintos envíos.
  - rank_source: proceso fuente.
  - MPI_Comm: comunicador.
  - status: estructura donde se indica el estado de la recepción.
    - status.MPI_SOURCE
    - status.MPI_TAG
    - status.MPI_ERROR

MPI: Primitivas Send y Recv en C++

```
int numero;
if(rank==0){
   /*Otro código*/
   numero=88;
   MPI_Send(&numero,1,MPI_INT,1,12,MPI_COMM_WORLD);
}
if(rank==1){
   /*Otro código*/
   MPI_Status status;
   MPI_Recv(&numero, 1, MPI_INT, 0, 12, MPI_COMM_WORLD, &status);
   cout<<"Se recibió "<<numero<<" desde proceso " <<status.MPI_SOURCE<<endl;
}
```

MPI: Primitivas Send y Recv en Python

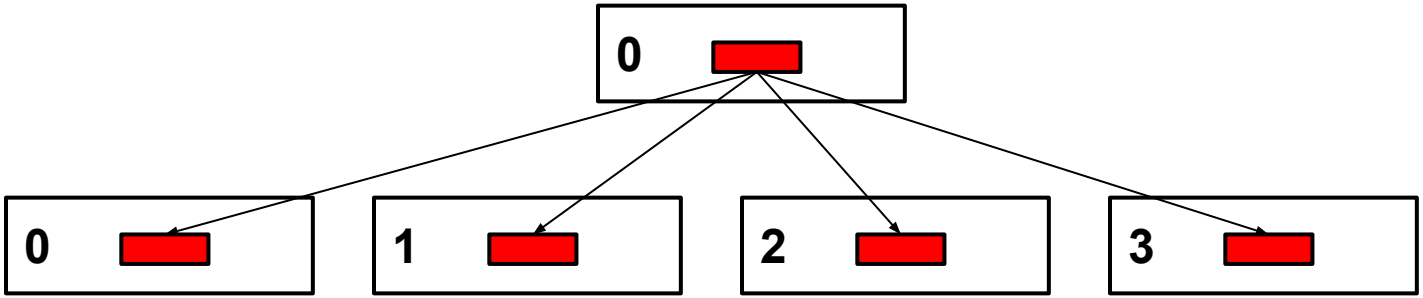
comm.send(dato,dest=i,tag=11)
dato=comm.recv(source=i,tag=11)

```
if rank==0:
    dato=88
    comm.send(dato,dest=1,tag=11)
if rank==1:
    dato=comm.recv(source=0,tag=11)
    print("Se recibió " + str(dato))
```

MPI: Primitiva bcast (broadcast)

- **C++: MPI_Bcast(void* buffer, count, MPI_Datatype, root, MPI_Comm comm);**
  - buffer: buffer donde están los datos a ser enviados o donde se depositarán los datos recibidos.

- **Python: dato_a_recibir = comm.bcast(dato_a_enviar, root=i)**



MPI: Primitiva bcast

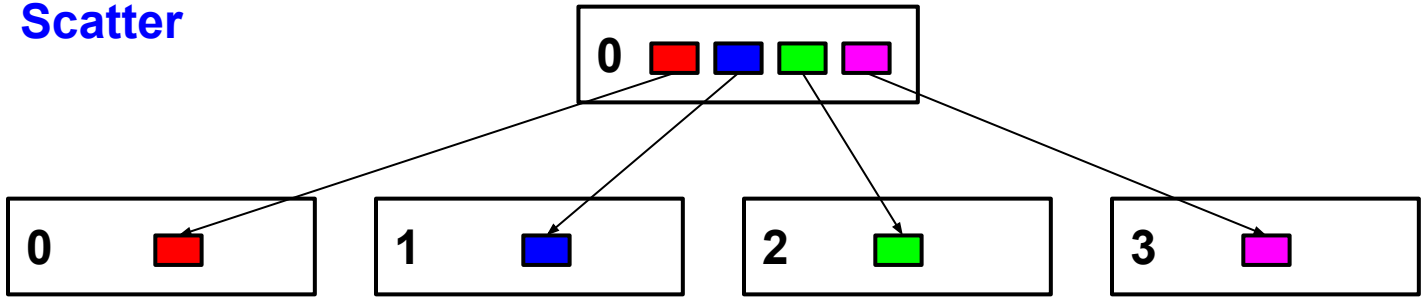
```
long double numero;
if(rank==0){
  cout<<"Ingrese el número: "<<endl;
  cin>>numero;
}

if(MPI_Bcast(&numero, 1, MPI_LONG_DOUBLE, 0, MPI_COMM_WORLD)!=MPI_SUCCESS){
  cout<<"Error ejecutando MPI_Bcast"<<endl;
  exit(1);
}
```

root

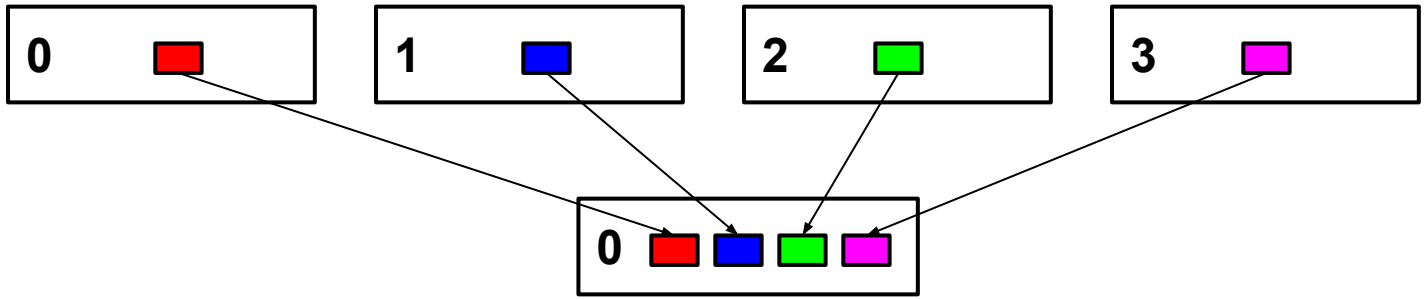
```
numero=0.0
if rank==0:
  numero=float(input("Ingrese el numero: "))

numero = comm.bcast(numero, root=0)
```

MPI: Primitivas scatter y gather (esparcir y reunir)

- **Scatter**



- **Gather**



MPI: Primitivas scatter y gather (esparcir y reunir)

C++

```
MPI_Scatter(void* sendbuf, sendcount, MPI_Datatype, void* recvbuf,
recvcount, MPI_Datatype, root, MPI_Comm);
MPI_Gather(void* sendbuf, sendcount, MPI_Datatype, void* recvbuf,
recvcount, MPI_Datatype, root, MPI_Comm);
```

- sendbuf: buffer con los datos a enviar.
- sendcount: cantidad de datos a enviar por proceso.
- Datatype: tipo de datos del buffer (MPI_INT, MPI_LONG).
- recvbuf: buffer de recepción.
- recvcount: cantidad de datos a recibir por proceso.
- root: proceso que esparce los datos (Scatter) o que los recolecta (Gather).

Python

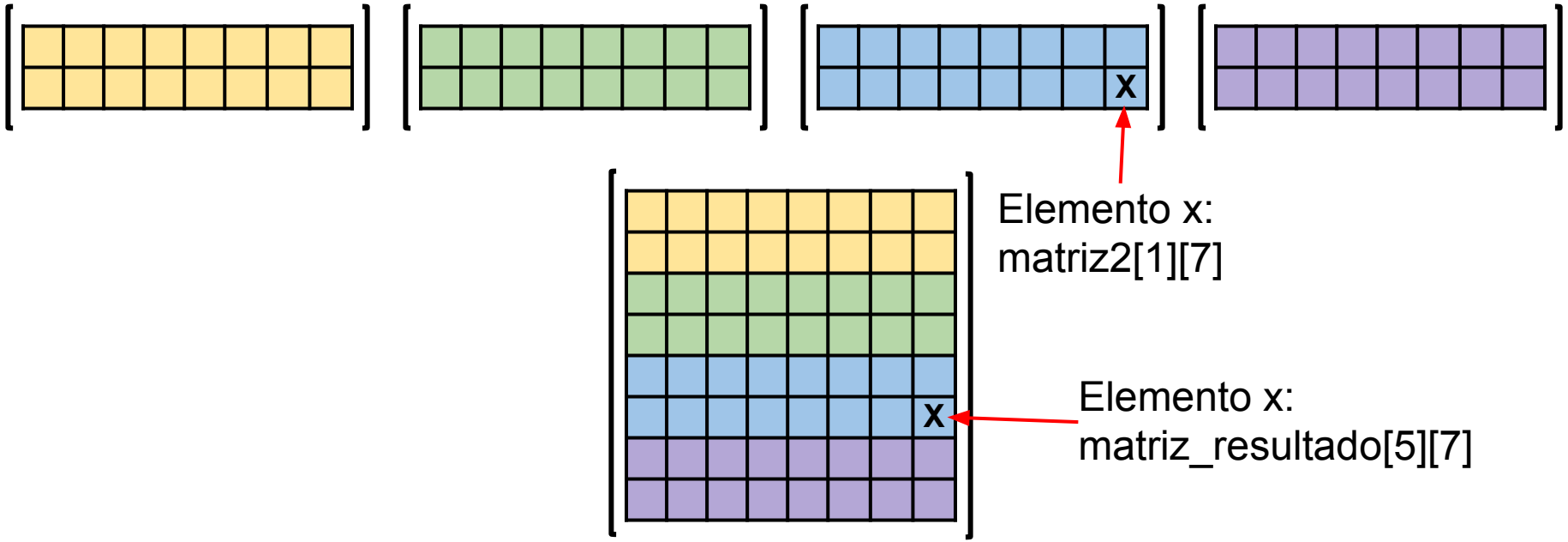
```
data = comm.scatter(arreglo, root=i)
arreglo = comm.gather(data,root=i)
```

MPI: Ejemplos de uso de gather con C++ y Python

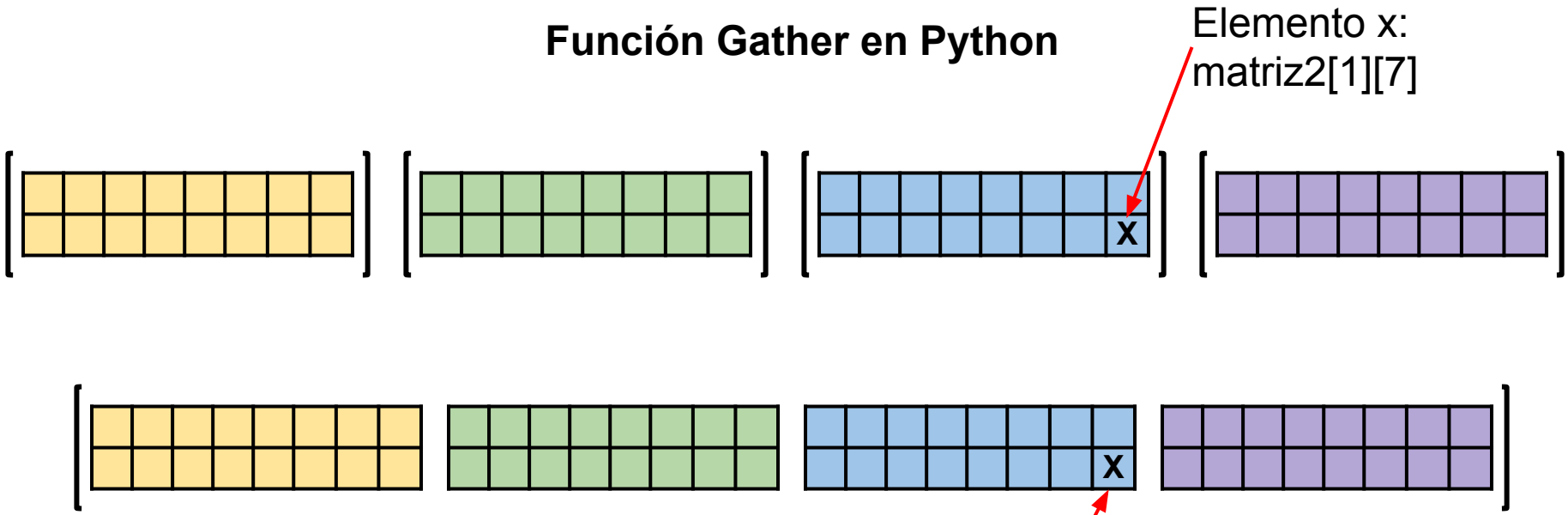
```
long double *matriz_resultados=new long double[size];
if(MPI_Gather(&ln_por_proceso, 1, MPI_LONG_DOUBLE, &matriz_resultados[0],
   1,MPI_LONG_DOUBLE, 0, MPI_COMM_WORLD)!=MPI_SUCCESS){
  cout<<"Error ejecutando MPI_Gather"<<endl;
  exit(1);
}
```

```
sumatoria=0.0
min=rank*cantidad_por_proceso
max=(rank+1)*cantidad_por_proceso
for i in range(min,max):
  sumatoria=sumatoria+(1.0/(2.0*float(i)+1.0))*(((numero-1.0)/-
(numero+1.0))**(2.0*float(i)+1.0))

resultados_list=comm.gather(sumatoria,root=0)
```

Función Gather en C++



Función Gather en Python



MPI: Otras Funciones

- Primitivas que permiten enviar datos de distinta longitud (MPI_Gatherv).
- Primitivas bloqueantes o síncronos (las vistas anteriormente son bloqueantes).
  - La siguiente instrucción no se ejecuta hasta que send o receive terminaron su trabajo.
- Primitivas no bloqueantes o asíncrono:
  - El programa sigue su ejecución mientras los datos se envían o reciben.
    - **Mayor performance.**
    - **Posibilidad de errores** (sobreescribir el buffer antes de que los datos terminen de enviarse).

Fuente: https://www.open-mpi.org/doc/v4.1/

Software de gestión

- En clusters compartidos:
  - Permiten a los usuario **solicitar recursos**.
  - Administran **colas de trabajo**.
    - Los administradores pueden definir políticas de prioridades.
  - Compila y copia los programas a todos los nodos.
  - Proveen herramientas para **monitorear** el estado de los trabajos y de los recursos.
- Usualmente poseen interfaz de usuario mediante consola de comandos a través de SSH.
- Algunas alternativas:
  - Slurm.
  - HTCondor.
  - PBS.

Administrador de colas de trabajo. Ejemplo: Slurm

- **sinfo**: muestra información sobre las colas de trabajo agendadas y el estado de los nodos.
- **squeue**: muestra los trabajos enviado a ejecución y el estado de los mismos.
  - Algunos estados:R (running), PD (pending), CA (cancelled), F (failed), TO (timeout), y NF (node failure).
- **sbatch**: ejecutar un trabajo en modo asíncrono.
  - Parámetros de ejecución (nodos, cantidad de CPUs, etc): Se pueden pasar por línea de comandos o utilizar un archivo (submit.job).
- **srun**: ejecutar un trabajo en modo síncrono.
- **scancel**: cancela un trabajo encolado (debe indicar el ID).
  - Debe averiguar el ID del trabajo (se obtienen con squeue).

https://slurm.schedmd.com/

Cloud Computing

- Pool de **recursos computacionales** **virtualizados** accesibles a través de una red o Internet.
  - Recursos computacionales:
    - Tiempo (en segundos) y poder (en GHz) de procesamiento.
    - Almacenamiento.
    - Servidores.
    - Máquinas virtuales
    - Software.
    - Plataformas de ejecución de aplicaciones.
- Primer antecedente: Amazon en 2006 ofrece servicios de infraestructura y almacenamiento.
  - Se suman Google y Microsoft.
  - Se comienzan a ofrecer otros tipos de servicios.
  - Constantemente se suman nuevas empresas y tipos de servicios.

Cloud Computing

- Características definidas por NIST[1] (National Institute of Standards Technology):
  - Autoservicio: Los clientes pueden proveerse unilateralmente (automáticamente) de servicios.
  - Servicio bajo demanda: Cada cliente es provisto con las capacidades computacionales que necesite.
  - Acceso a través de la red disponible por el cliente.
  - Acceso ubicuo: Podemos acceder a nuestra máquina virtual o aplicación en cualquier parte del mundo con acceso a Internet.
  - Modelo multi-cliente: Los proveedores ofrecen servicios a múltiples clientes.
  - Los recursos pueden ser provistos o liberados en forma rápida y dinámica.
  - Los servicios son medidos.

[1] NIST Special Publication 800-145: The NIST Definition of Cloud Computing

Cloud Computing

- Arquitectura de 4 capas:
  - Hardware.
  - Infraestructura: Capa de middleware de virtualización. Provee recursos virtualizados, cómo procesamiento, almacenamiento, etc. (por ejemplo, una máquina virtual).
    - Provee el servicio de Infrastructure as a Service (IaaS)
  - Plataforma: Plataforma para ejecutar aplicaciones del cliente (similar a una máquina con un sistema operativo sobre la cual las aplicaciones pueden correr). Provee alta abstracción de los recursos.
    - Provee el servicio de Platform as a Service (PaaS)
  - Aplicación: Software provisto como servicio.
    - Provee el servicio de Software as a Service (SaaS).
- Las capas pueden ser del mismo o de diferentes proveedores.

Cloud Computing

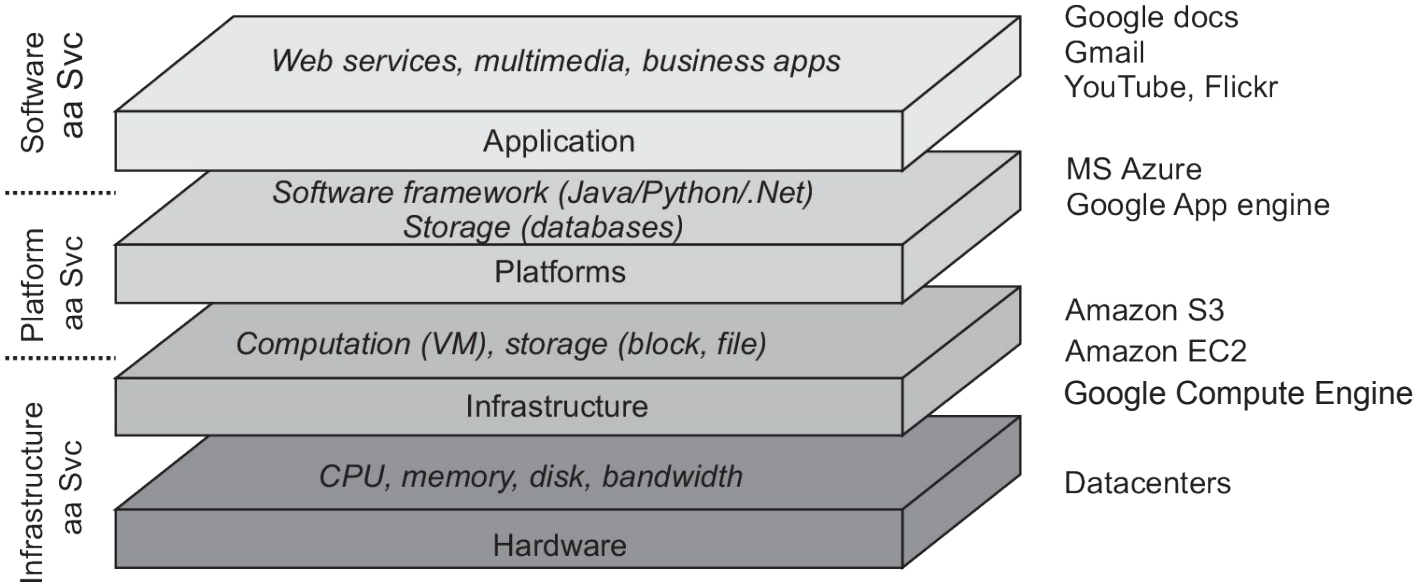


Figura obtenida de van Steen and Tanenbaum, "Distributed Systems", Third edition, v 3.03 (2020), pag 30.

Cloud Computing

Modelos básicos de servicios y arquitectura:

- **Software as a Service (SaaS)**: El cliente puede ejecutar las **aplicaciones del proveedor** sobre la infraestructura Cloud.
- **Platform as a Service (PaaS)**: El cliente puede ejecutar las **aplicaciones del cliente** sobre la infraestructura Cloud (las librerías, servicios, lenguajes, etc. deben ser soportados por el proveedor).
- **Infrastructure as a Service (IaaS)**: El proveedor ofrece recursos computacionales virtualizados (poder de procesamiento, almacenamiento, redes, otros dispositivos específicos, etc.). El cliente puede instalar sistemas operativos y aplicaciones que desee.

Cloud Computing

Otros servicios:

- DaaS (Database as a Service)
- FaaS (Framework as a Service)
- DaaS (Desktop as a Service)
- CaaS (Contenedores como servicios)
- Virtual Private Cloud (VPC): recursos + VPN + IP.
- Device Farm: Dispositivos móviles reales para probar código en ellos.

Frameworks para Cloud Privados

- Eucalyptus (compatible con AWS)
- OpenNebula
- OpenStack
- VMware vSphere

Cloud Computing

- Ventajas de Cloud Computing:
  - **Robustez**: Los proveedores de servicios se encargan de la instalación, mantenimiento y actualización de software y hardware, respaldo y seguridad de los datos y aplicaciones almacenadas y calidad de servicio.
  - **Disponibilidad**: El proveedor asegura la disponibilidad deseada por el cliente (por ejemplo: 24 hs todo el año), utilizando infraestructura de respaldo.
    - Tolerancia a fallos: mecanismos de failover, recursos distribuidos en diferentes partes del mundo, etc.
  - **Escalabilidad y elasticidad**: Los recursos pueden escalar para adaptarse a las demandas del cliente de manera automática (para el cliente los recursos parecen ilimitados)

Ejemplos de Cloud Computing

Proveedores de servicios de Cloud Computing:

- Google Cloud Platform: (https://cloud.google.com/):
  - PaaS: App Engine (algunos recursos gratis).
  - IaaS: Compute Engine (algunos recursos gratis).
- Amazon Web Services:
  - PaaS: AWS Lambda (algunos recursos gratis).
  - IaaS:
    - EC2 (Elastic Compute Cloud).
    - Amazon S3 (Simple Storage Service)
- Windows Azure:
  - PaaS: App Service.
  - IaaS: virtual machines.

Ejemplos de Cloud Computing

IaaS Cloud computing: computación

- Ejemplo: Amazon EC2 (Elastic Compute Cloud).
- Ejemplo: Google Compute Engine.

- Se proveen varios tipos de **máquinas virtuales preconfiguradas**.
  - Amazon les llama AMI (Amazon Machine Image). Google les llama Instancias.
- Los tipos de imágenes se diferencian en:
  - CPU, número y tipos de núcleos, y tipo de GPU.
  - Memoria.
  - Almacenamiento no volátil.
  - Plataforma (32 bits o 64 bits).
  - Networking: Capacidades de red (ancho de banda).
  - Aplicaciones y herramientas preinstaladas (Base de datos, servidores web,)

Ejemplos de Cloud Computing

IaaS Cloud computing: almacenamiento

- Ejemplo: Elastic Block Store (Amazon EBS).
  - Puede ser utilizado como un disco duro (virtual).

- Google Cloud storage: Almacenamiento de objetos.
- Persistent disk: Almacenamiento como un disco duro (virtual).

- Ejemplo de instancia gratuita: Instancia **e2-micro** de Google Cloud Platform (instancia gratuita sin necesidad de indicar tarjeta de crédito).
  - Procesador Intel Xeon 2.20 GHz 64 bits de 4 núcleos (dos núcleos físicos, 2 hilos por núcleo), 16 GB RAM, 114 GB (características con mucha variación).
  - Subir archivos (almacenamiento persistente).
  - Interfaz a través de consola de comandos Linux.
  - En ejecución solo cuando está la interfaz web (cliente) corriendo.

Ejemplos de máquinas virtuales de Google Cloud Computing

E2-micro: 2 núcleos; 8 GB de memoria; 10 GB disco (7.82 USD por mes).
C2-standard-60: 60 núcleos; 240 GB de memoria; 128 discos; ancho de banda de salida: 32 Gbps (1829.62 USD por mes).

Ejemplos de máquinas virtuales (IaaS) de Google

Máquinas de uso general: https://cloud.google.com/compute/docs/general-purpose-machines
Máquinas optimizada para procesamiento: https://cloud.google.com/compute/docs/compute-optimized-machines
Máquinas con optimización de memoria: https://cloud.google.com/compute/docs/memory-optimized-machines
Calculadora de precios: https://cloud.google.com/compute/vm-instance-pricing
Calculadora de precios: https://cloud.google.com/products/calculator
Productos gratuitos: https://cloud.google.com/free/

Grid Computing

- Infraestructura descentralizada para compartir recursos geográficos distribuidos y diversos (heterogéneos) que son propiedad de diferentes organizaciones y compartidos por las mismas.
  - Supercomputación.
  - Sistemas de almacenamiento.
  - Fuentes de datos.
  - Dispositivos especializados.
- Objetivos:
  - Resolver problemas de gran escala que requieren elevado poder de procesamiento o almacenamiento.

Ejemplos de Grids

- SETI@home: Grid construida para buscar patrones de vida extraterrestre.
  - Lanzado por la universidad de California 1999 hasta 2020.
  - Formada por computadoras hogareñas de todo el mundo que sumaban poder de cómputo voluntariamente.
  - Analizaba datos de radiotelescopios.
  - Llegó a tener 1.8 millones de voluntarios activos y 278832 computadoras activas al mismo tiempo. Alcanzó 1 PFlops (Clementina XXI posee 3.88 PFLOPs y las últimas máquinas del Top500 rondan los 2 PFlops).
- GIMPS: Grid destinada a encontrar números primos de Mersenne (https://www.mersenne.org/download/).

Ejemplos de Grids

- Folding@home. Grid para simulaciones en medicina. Universidad de Stanford. (https://foldingathome.org/).
  - Comenzó realizando simulaciones de plegamiento proteico.
  - Actualmente realiza simulaciones sobre varias enfermedades.
  - Durante la pandemia de COVID creó un proyecto para realizar simulaciones que contó con gran apoyo a nivel mundial.
  - Alcanzó una potencia de cálculo pico de **2.4 HexaFLOP** en 2020. (superó a todas las supercomputadoras del Top500 de ese momento).
- EGI (European Grid Infrastructure):
  - Analiza datos producidos por el Gran Colisionador de Hadrones en el CERN entre otros.
  - Formado por computadoras de centros de datos e instituciones académicas.,

Paralelismo en el software: Granularidad

Nivel	Plataforma típica	Tipo Memoria	Comunicación
Procesos independientes (programas diferentes)	Multinúcleo, multicomputadoras.	Distribuida	Mensajes
Procesos de una misma tarea.	Multinúcleo, multicomputadoras.	Distribuida	Mensajes (MPI), RPC.
Hilos.	Multinúcleo, multihilo simultáneo, GPU.	Compartida	Variables compartidas
Lazos no recursivos, SIMD.	Procesadores SIMD, vectoriales, GPU.	Compartida	Variables compartidas
Instrucción	Pipeline, superescalar, fuera de orden	Compartida	Variables compartidas

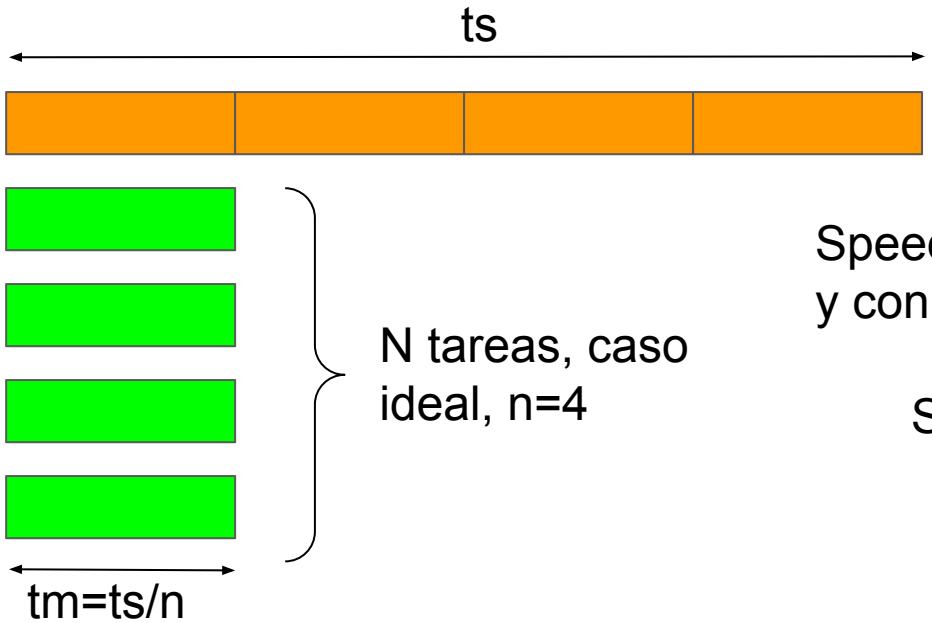
Paralelismo en el software: Granularidad

Nivel	Quién decide	Herramienta
Procesos independientes (programas diferentes)	SO, programador.	
Procesos de una misma tarea.	Programador	MPI, RPC.
Hilos.	Programador, compilador, SO.	CUDA, OpenCL, OpenMP, librerías multihilos
Lazos no recursivos, SIMD.	Compilador (lazos), programador (SIMD, GPU).	CUDA, OpenCL, instrucc. SIMD.
Instrucción	Procesador, Compilador (VLIW).	

Medidas de performance: Speedup

Speedup ideal: S(n)= $\frac{ts}{tm}$

ts=Tiempo total n tareas ejecución secuencial.
tm=Tiempo de cada tarea ejecutándose en paralelo.



Speedup ideal, sin overhead de comunicación y con código 100% paralelizable

$$\text{Speedup: } S(n)= \frac{ts}{tm} = \frac{ts}{ts/n} = n$$

Speedup Ideal

Ejecución paralela con código no paralelizable (serial)

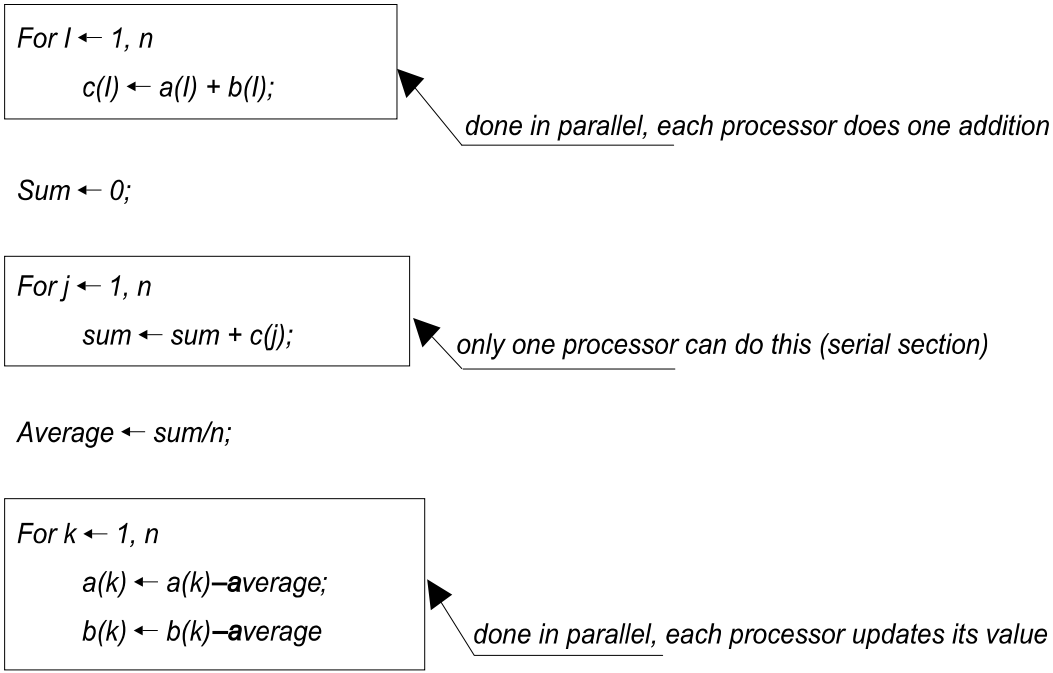


Figura obtenida de: Hesham and Mostafa, “Advanced Computer Architecture and Parallel Processing”, 1° edición, página 53

Ejecución paralela con código no paralelizable

- Para analizar el efecto de la fracción de código no paralelizable, debemos considerar el **tipo de problema** a paralelizar.
  - Modelo de **Amdahl** o problema de **tamaño fijo**.
    - Problemas donde el tiempo para resolver el problema es el parámetro crítico.
  - Modelo de **Gustafson-Barsis** o problema de **tamaño variable proporcional** a la cantidad de tareas que pueden ejecutarse en paralelo.
    - Problemas donde se adapta la precisión del resultado o tamaño de los modelos a los recursos disponibles.
  - Modelo de **Sun y Ni** o de **memoria total limitada**.
    - Problemas con grandes cantidades de datos.

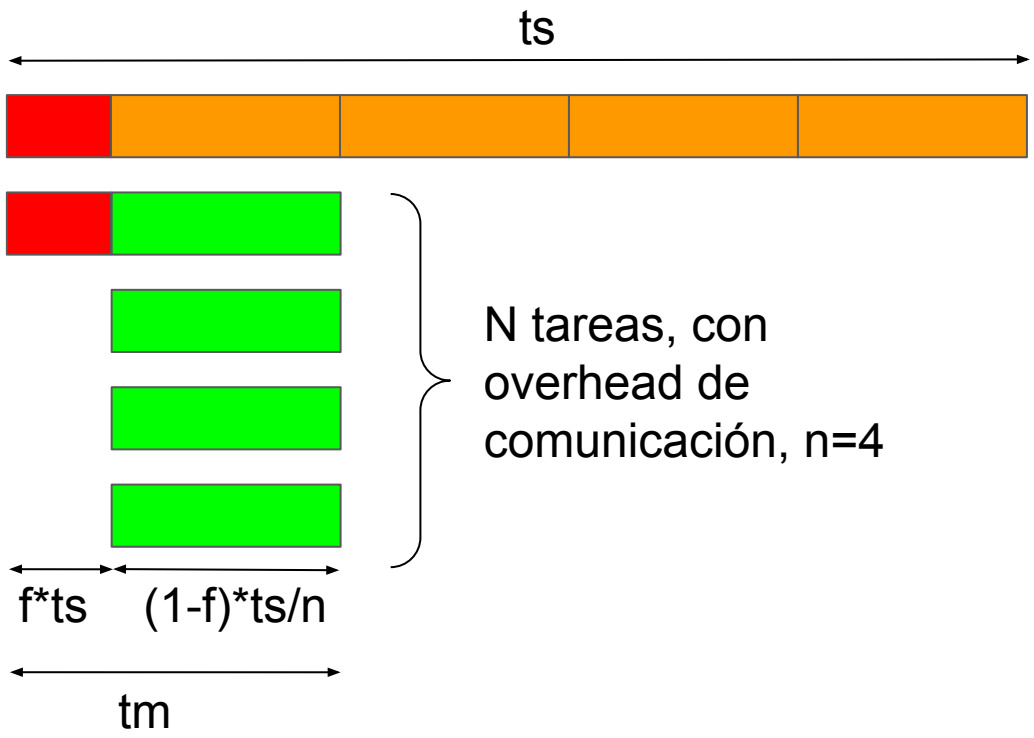
No confundir modelo de Amdahl con Ley de Amdahl. La Ley de Amdahl (“La mejora global que se consigue al mejorar un componente de un sistema de cómputo depende de la fracción de tiempo que se use dicho componente”) está planteada para problemas que siguen el modelo de Amdahl.

Ejecución paralela con código no paralelizable según modelo de Amdahl

- Sin importar el overhead

$$tm = f * ts + (1-f) * \frac{ts}{n}$$

f: Fracción de código no paralelizable (serial)



Ejecución paralela con código no paralelizable según modelo de Amdahl

- Sin importar el overhead de comunicación

$$tm = f * ts + (1-f) * \frac{ts}{n}$$

Fracción de código no paralelizable (serial)

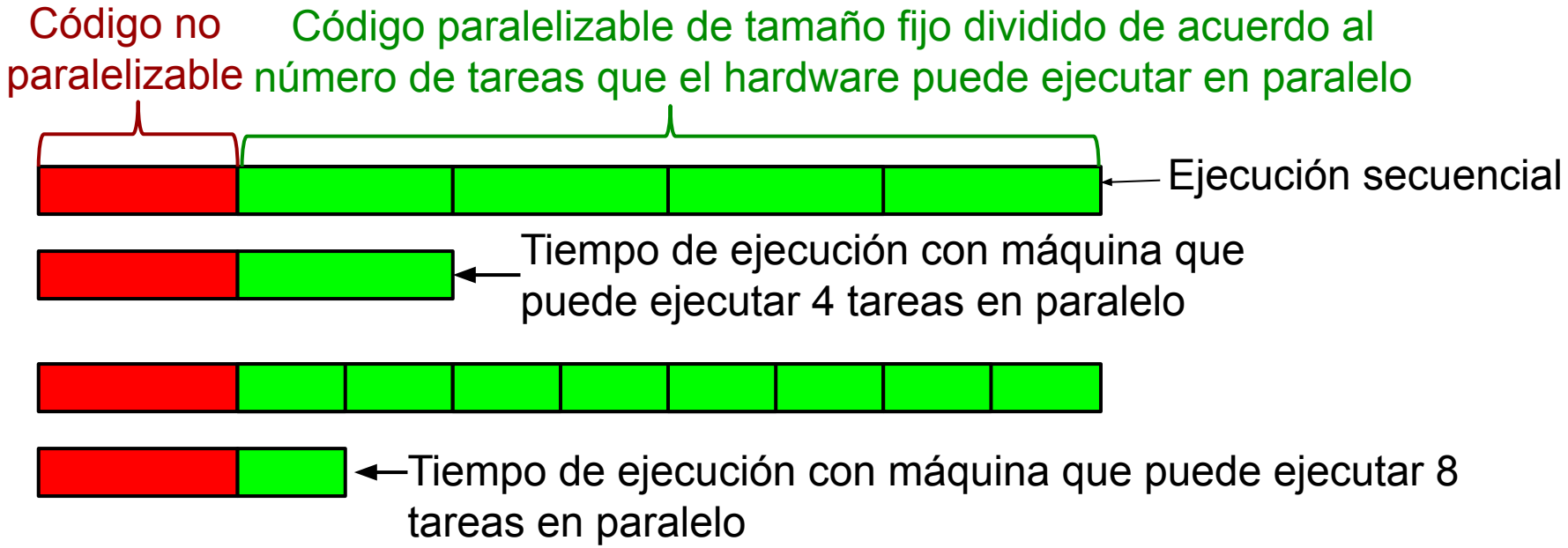
$$S = \frac{ts}{f * ts + \frac{(1-f) * ts}{n}} = \frac{n}{1 + (n-1) * f}$$

f = 1 (ninguna parte del código es paralelizable) ⇒ S = 1

$n \rightarrow \infty \Rightarrow S = 1/f$ ← **Sin importar la arquitectura ni n, el S máximo está limitado por la porción de código no paralelizable**

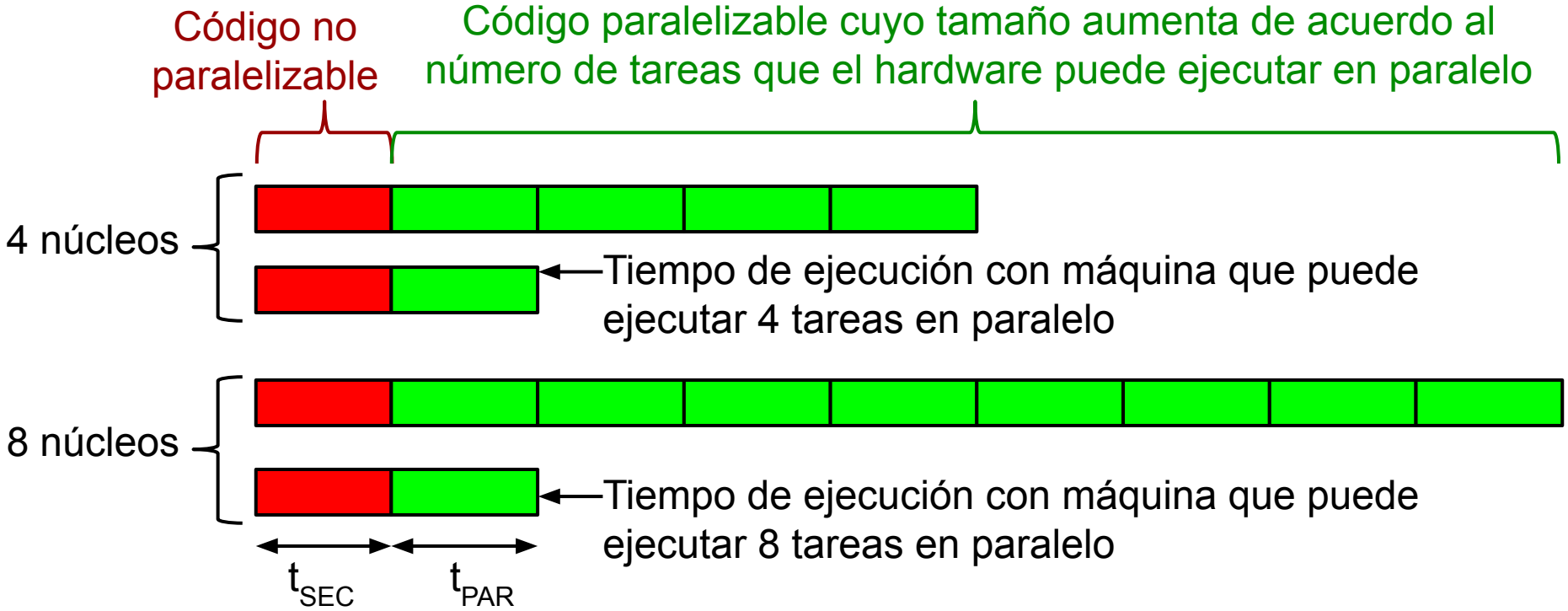
Speedup según Ley de Amdahl

- Modelo de **Amdahl** supone que el tiempo total o **tamaño total del problema es fijo**.
  - Si aumenta n, disminuye el tiempo de ejecución en cada procesador.
  - Ejemplo: Problemas donde se busca el menor tiempo de ejecución.



Speedup según Gustafson-Barsis

- Supone que los tiempos t_{SEC} y t_{PAR} son constantes, y el tiempo total o **tamaño del problema aumenta con el valor de n**.



Speedup: Ley de Gustafson-Barsis

$$S = \frac{t_{SEC} + t_{PAR} * n}{t_{SEC} + t_{PAR}}$$

(Ver figura filmina anterior)

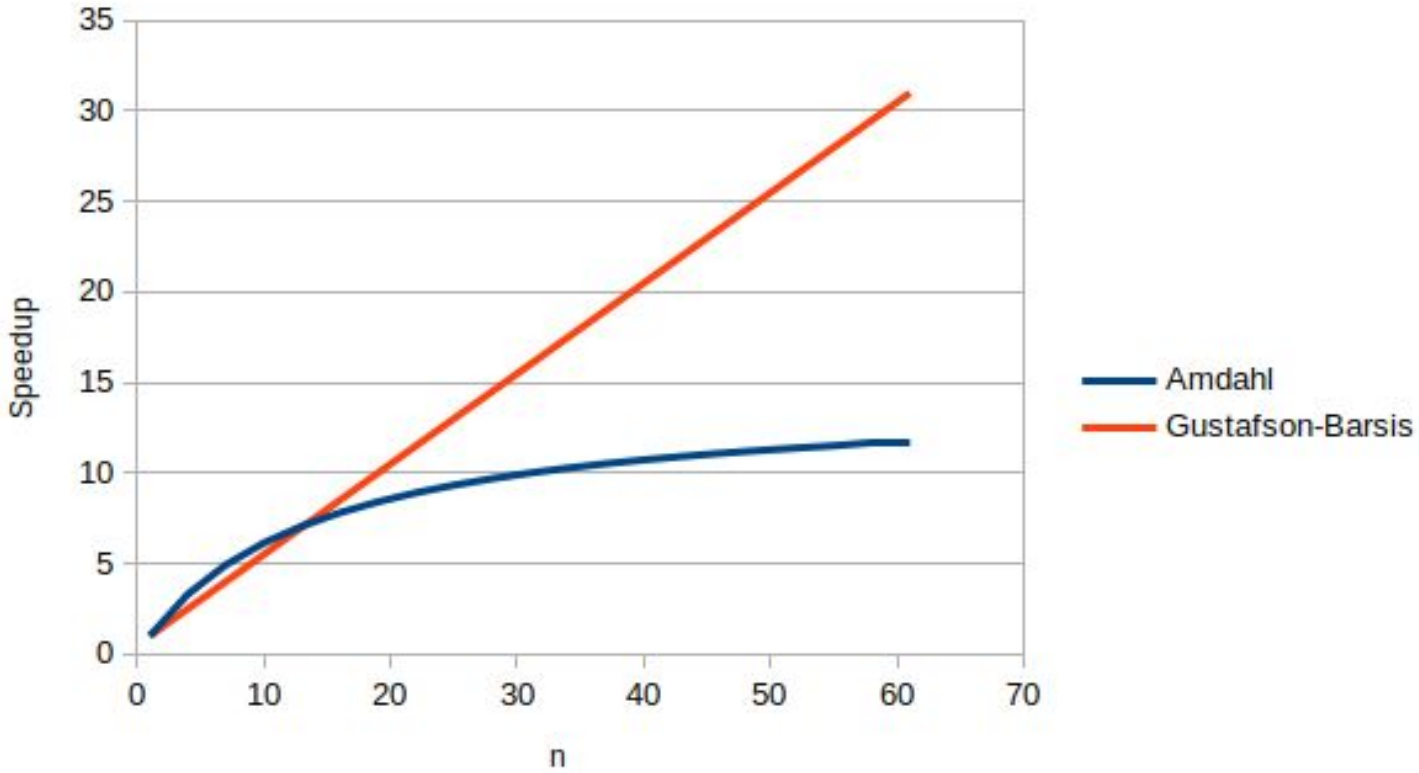
$$a = \frac{t_{SEC}}{t_{SEC} + t_{PAR}} \Rightarrow (1 - a) = 1 - \frac{t_{SEC}}{t_{SEC} + t_{PAR}} = \frac{t_{SEC} + t_{PAR} - t_{SEC}}{t_{SEC} + t_{PAR}} = \frac{t_{PAR}}{t_{SEC} + t_{PAR}}$$

$$S = \frac{t_{SEC} + t_{PAR} * n}{t_{SEC} + t_{PAR}} = \frac{t_{SEC}}{t_{SEC} + t_{PAR}} + \frac{t_{PAR} * n}{t_{SEC} + t_{PAR}} = a + (1 - a)*n$$

a<1

Si $n \rightarrow \infty \Rightarrow S \rightarrow \infty$

f=0,07


Speedup
35
30
25
20
15
10
5
0
0
10
20
30
40
50
60
70
n
Amdahl
Gustafson-Barsis

Speedup según Gustafson-Barsis

- La **Ley de Amdahl** indica que **el speedup tiene un límite superior** para **problemas de tamaño fijo**.
  - Conclusión muy negativa, ya que indica que “no vale la pena” incrementar la cantidad de procesadores dado un valor de f para conseguir mayores speedup.
- La **Ley de Gustafson-Barsis** indica que **el speedup no posee límite** superior si el **tamaño del problema crece linealmente** con el número de procesadores.
  - **Conclusión muy positiva, ya que indica que si vale la pena incrementar el número de procesadores para conseguir mayores speedup**.
- Ambos suponen modelos ideales, pero los problemas reales se acercan más al modelo de la **Ley de Gustafson** ya que:
  - Los problemas requieren la mayor precisión posible (mayor número de decimales, menor tamaño de grilla), lo cual se logra incrementando n.
  - El tamaño de los problemas se adapta al n disponible, y no al revés.

Bibliografía:

- William Stallings, "Computer Organization and Architecture”, 10º edición, editorial Pearson, año 2016.
- Tanenbaum and Bos, “Modern Operating Systems”, 4° edición, editorial Pearson, año 2015.
- Kai Hwang, “Advanced Computer Architecture, Parallelism, Scalability, Programmability”, 2º edición, editorial Mc Graw-Hill, año 2011.
- Hesham and Mostafa, “Advanced Computer Architecture and Parallel Processing”, 1° edición, editorial Wiley, año 2005.
- ARM, "ARM Cortex-A Series Programmer’s Guide for ARMv8-A", Version 1.0, año 2015
- MPICH Documentation (https://www.mpich.org/documentation/guides/)
- mpi4py Tutorial (https://mpi4py.readthedocs.io/en/stable/tutorial.html)

Bibliografía:

- Buyya, Yeo, Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities", 2008.
- NITS (National Institute of Standards Technology), “The NIST Definition of Cloud Computing”.
- Bhaskar Prasad Rimal, Eunmi Choi, Ian Lumb, "A Taxonomy and Survey of Cloud Computing Systems".