

El Bueno, el Malo y el Feo

Mejorando la Eficiencia y Calidad del Software Paralelo

Ricardo Medel

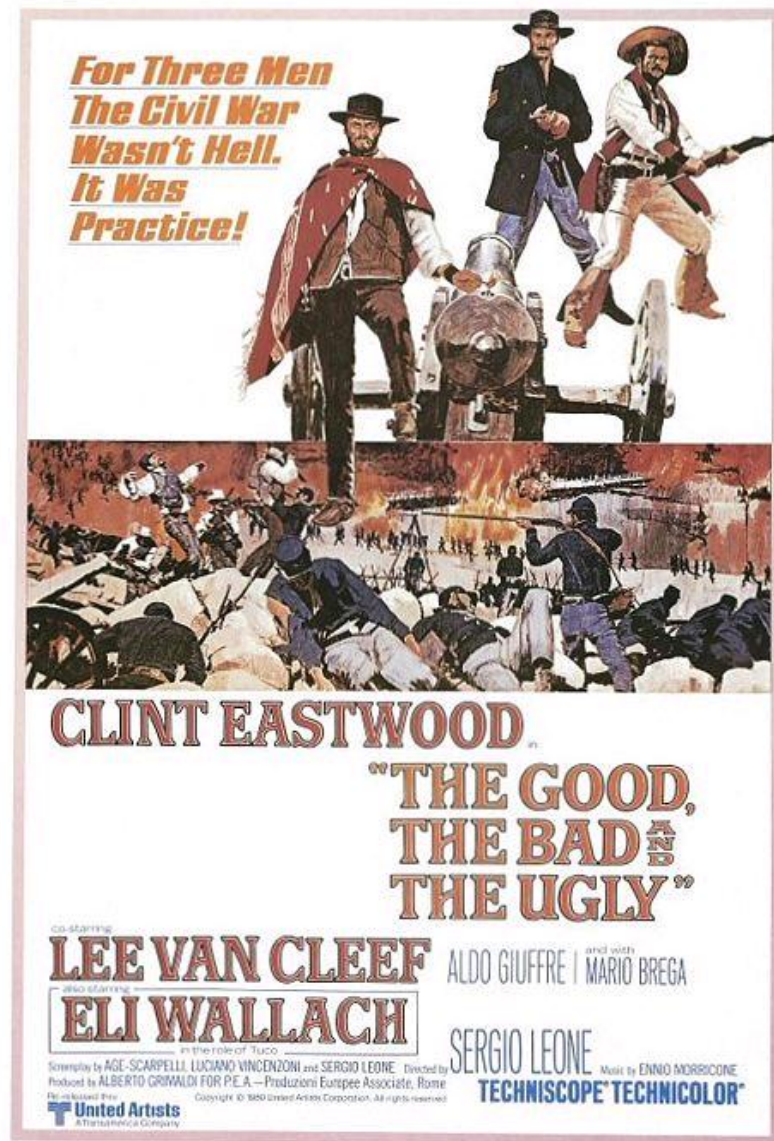
Argentina Systems & Tools
SSG-DPD

Basado en un gui3n de

Eric W. Moore - Senior Technical Consulting Engineer



La película



Intel Confidential

Optimization
Notice

Clasificación de computadoras

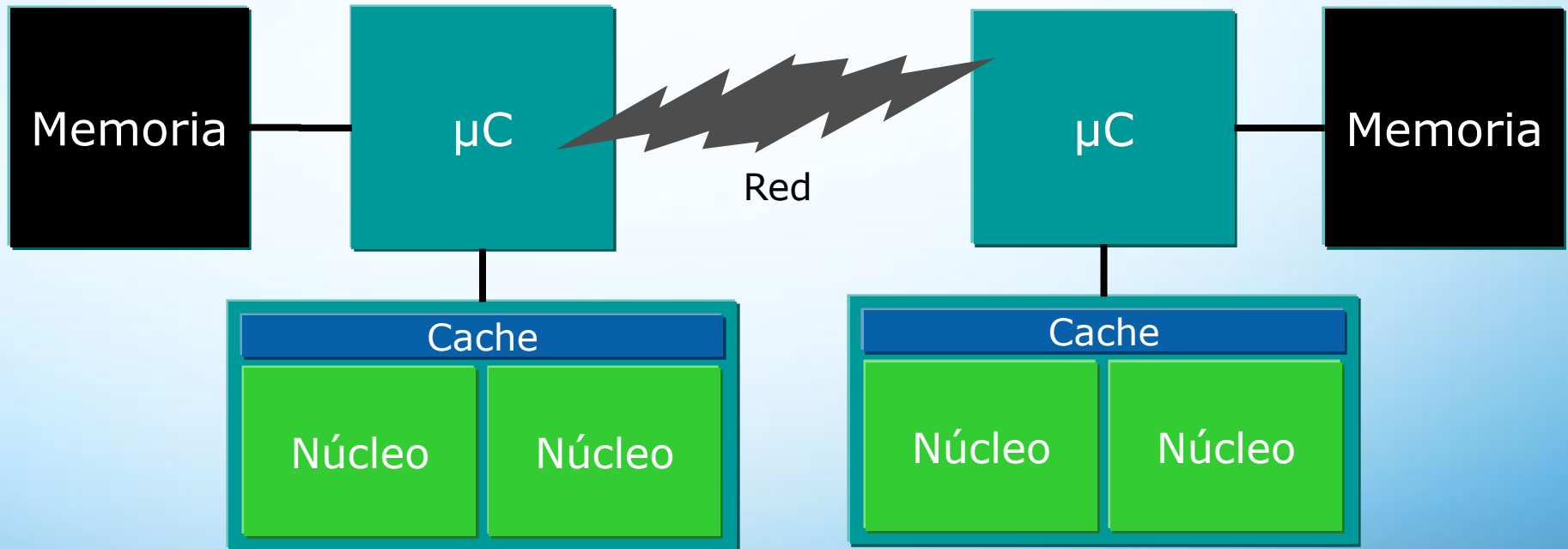
- Taxonomía de Flynn (Instrucciones/Datos)
 - SISD
 - MIMD (SPMD)
 - SIMD
 - MISD
- Tipo de Acceso a Memoria
 - Multiprocesamiento Simétrico (SMP)
 - Multiprocesamiento Asimétrico (AMP)
 - Acceso No-Uniforme a Memoria (NUMA)

NUMA: Acceso No-Uniforme a Memoria

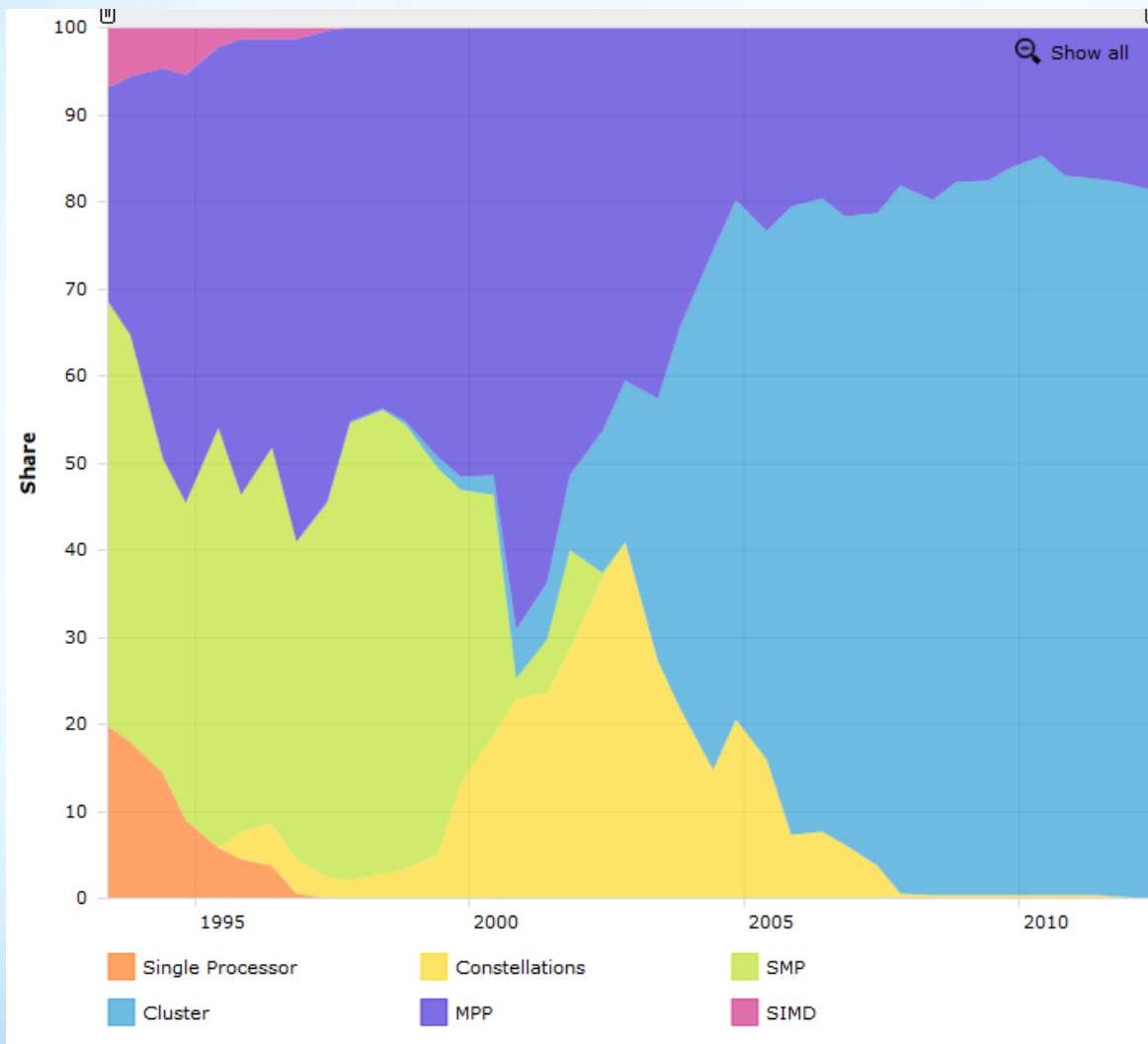
- Non-Uniform Memory Access
- Múltiples CPUs
- Múltiples Memorias
- Único o Múltiples Sist. Op.

• Ejemplos

- GPGPU
- Clústeres HPC
- Grid

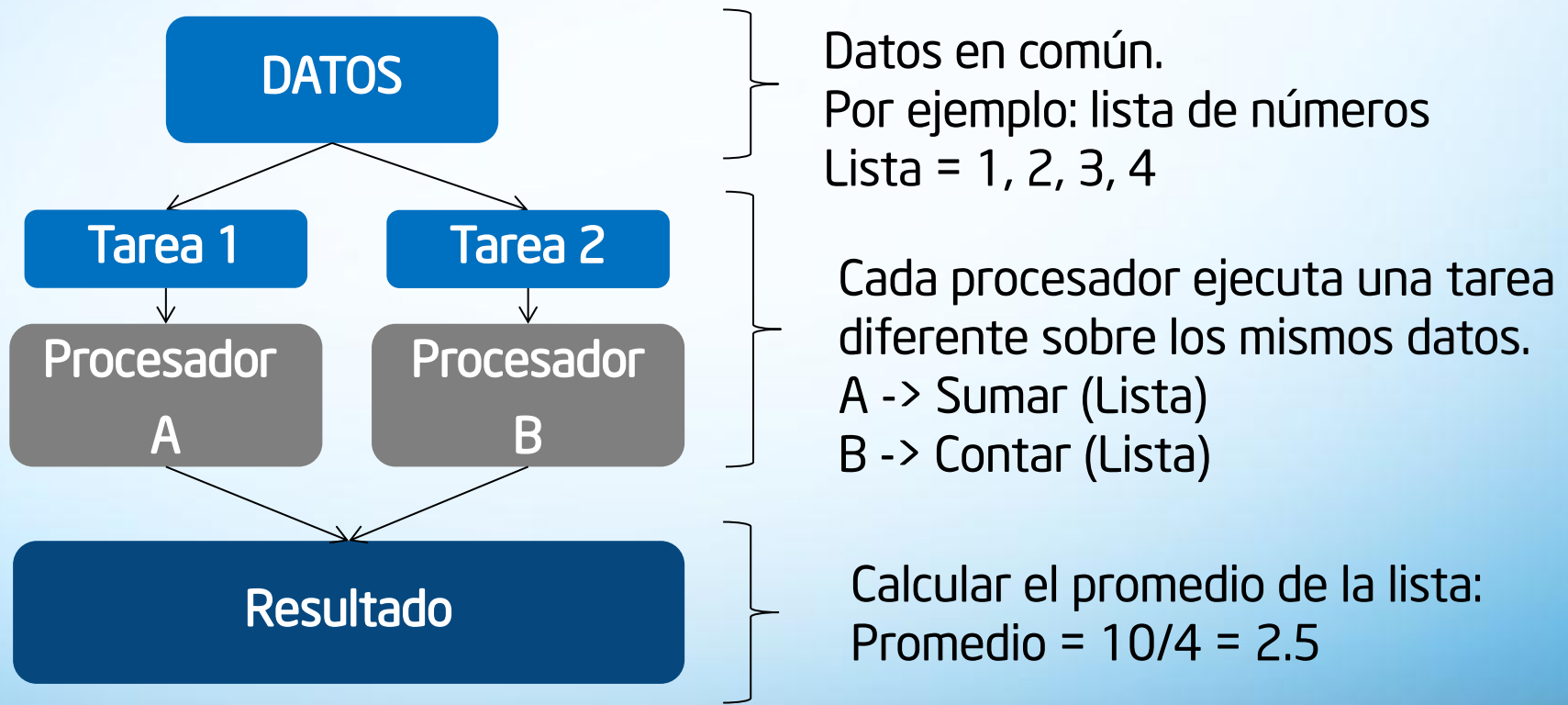


Clústeres (NUMA) en el Top500.org



Paralelismo de Tareas

Task parallelism es una forma de paralelismo que se enfoca en la distribución de diferentes tareas entre diferentes procesadores paralelos.



Pipelining

Pipelining es el proceso en el cual los elementos que procesan un conjunto de datos están conectados en serie.

Un ejemplo de este concepto es la línea de ensamblado:



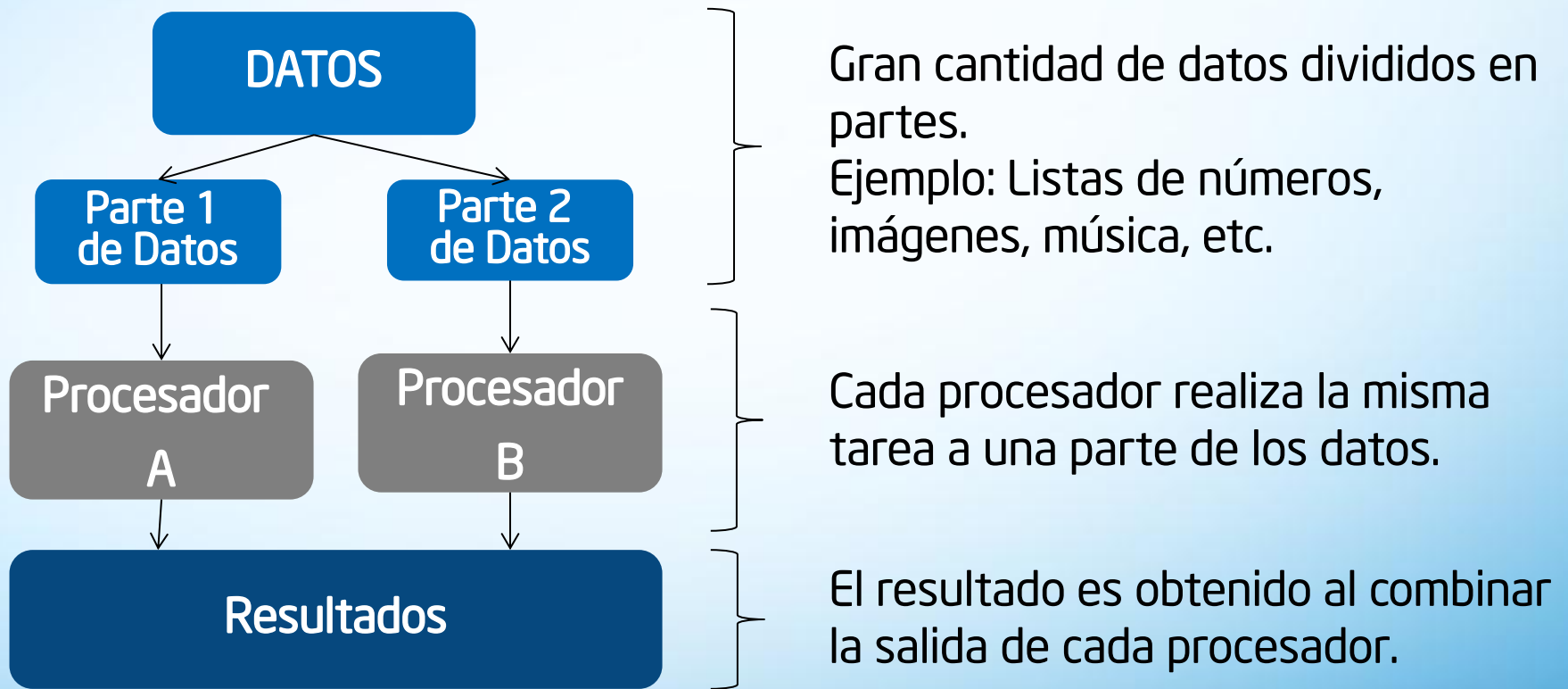
Ejecutemos con dos entradas: X (entrada 1) e Y (entrada 2)

	Tarea 1 (t=3 seg)	Tarea 2 (t=2 seg)	Tarea 3 (t=2 seg)
Ciclo 1	X		
Ciclo 2	Y	X	
Ciclo 3		Y	X
Ciclo 4			Y

- Primera salida toma 7seg
- Otra salida cada 3 seg
- La demora está dada por la tarea más larga

Paralelismo de Datos

Data parallelism es una forma de paralelismo en la cual los datos son divididos entre múltiples procesos que realizan la misma operación.



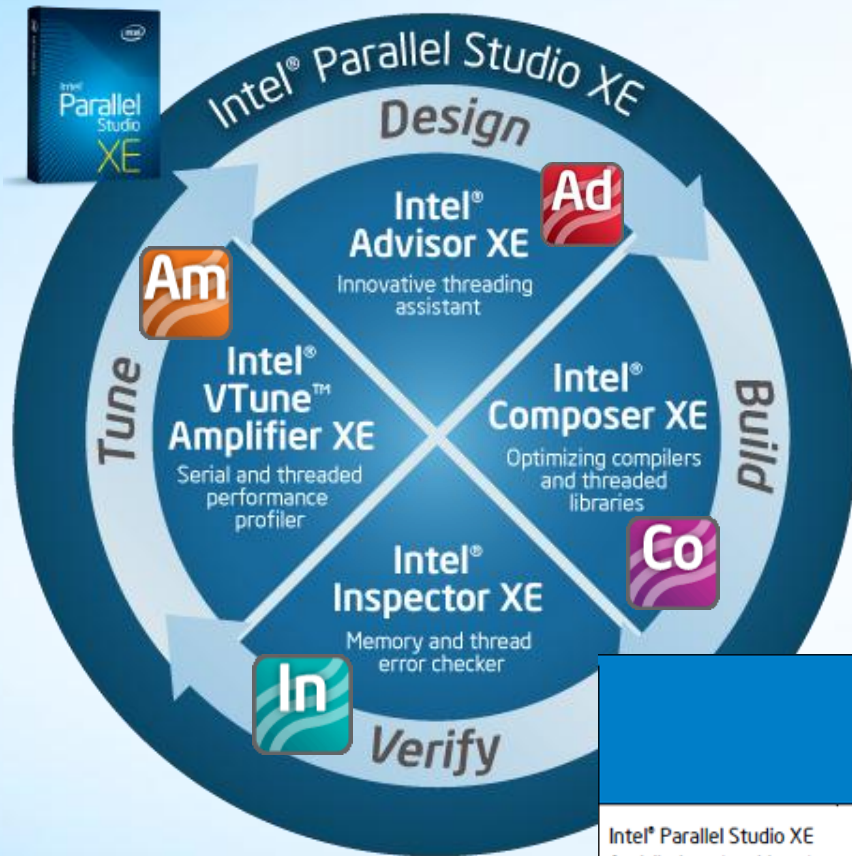
Modelos de Programación

- Hilos (Threads)
 - Cada flujo de control es un “hilo”
 - Comparten el espacio de memoria
- Comunicación Entre Procesos
 - Cada flujo de control es un proceso (del Sist. Op.)
 - *nix
- Distribuído
 - Pasaje de Mensajes (MPI: Message Passing Interface)
 - Espacio de Memoria Particionado

Diseño de un Programa Paralelo

1. Entender el **problema** para verificar que pueda ser resuelto con un programa paralelo.
2. Identificar en los algoritmos las **dependencias de datos** y si es posible eliminarlas.
3. Identificar posibles **algoritmos alternativos** que puedan ser paralelizados.
4. Identificar los **hotspots** del programa. Secciones donde la mayor parte del trabajo es realizado. Reditúa más paralelizar los estas secciones y no las que usan poca CPU.
5. Identificar los **cuellos de botella** del programa. Secciones donde uno o varios procesos esperan al resto. Evitarlos o minimizarlos.

Intel® Parallel Studio XE



Un conjunto de herramientas para todo el ciclo de vida del desarrollo de software.

- **ANÁLISIS & DISEÑO**
- **CODIFICACIÓN & TESTING**
- **VERIFICACIÓN**
- **PUESTA A PUNTO**

	Intel® C / C++ Compiler	Intel® Fortran Compiler	Intel® Integrated Performance Primitives Library	Intel® Math Kernel Library	Intel® Cilk™ Plus	Intel® Threading Building Blocks	Intel® Inspector XE	Intel® VTune™ Amplifier XE	Static Security Analysis
Intel® Parallel Studio XE for Windows* or Linux*	•	•	•	•	•	•	•	•	•
Intel® C++ Studio XE for Windows or Linux	•		•	•	•	•	•	•	•
Intel® Fortran Studio XE for Windows or Linux		•		•			•	•	•

Buscando Hotspots

Usando  Intel® VTune Amplifier XE 2011

Elapsed Time: 13.205s

Total Thread Count: 1
CPU Time: 13.201s
Paused Time: 0s
Frame Count: 0

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function CPU Time

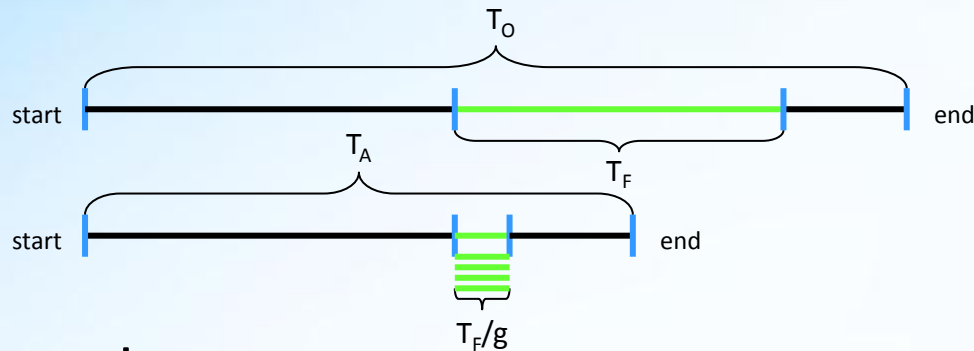
fctr 6.146s
ffld 1.950s
gf 1.249s
sin.N 0.858s
gx 0.610s
[Others] 2.389s

Line	Source	CPU Time	☆
4695	pj= ip[j]-1; /*rest one coz ip start with 1 and sc		
4696	arj= scm[pj];	0.010s	
4697	a[j+r*ndim]= arj;	0.010s	
4698	scm[pj]= scm[j];		
4699	jpl= j+1;		
4700			
4701	for(i = jpl; i < n; i++)	0.910s	
4702	scm[i] -= a[i+j*ndim]* arj;	5.166s	



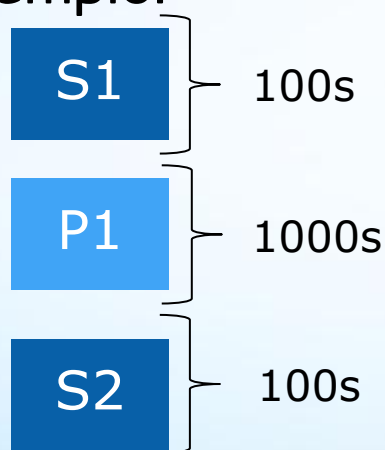
Ejemplo de uso de Intel VTune Amplifier
Digging for Gold (en C++)

Ley de Amdahl



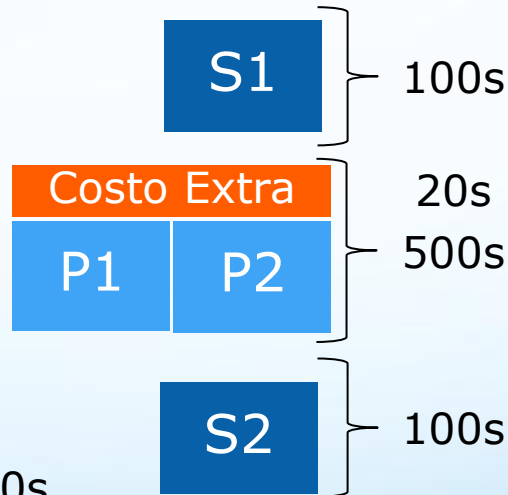
Un programa tiene secciones paralelizables y secciones en serie

Ejemplo:



Tiempo Total: 1200s

2 partes serie y
1 parte paralela



Tiempo Total: 720s
Aceleración = $1200/720s$
= 1.66

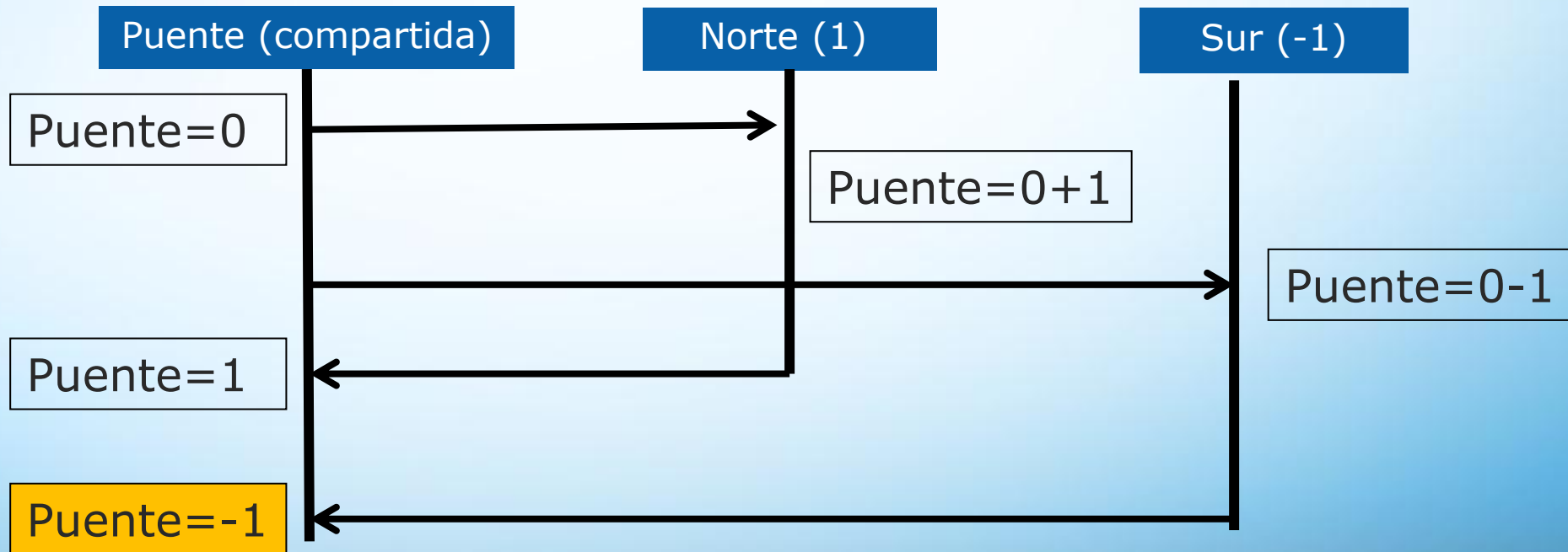


Tiempo Total: 490s
Aceleración = $1200/490s$
= 2.45

Condición de Carrera

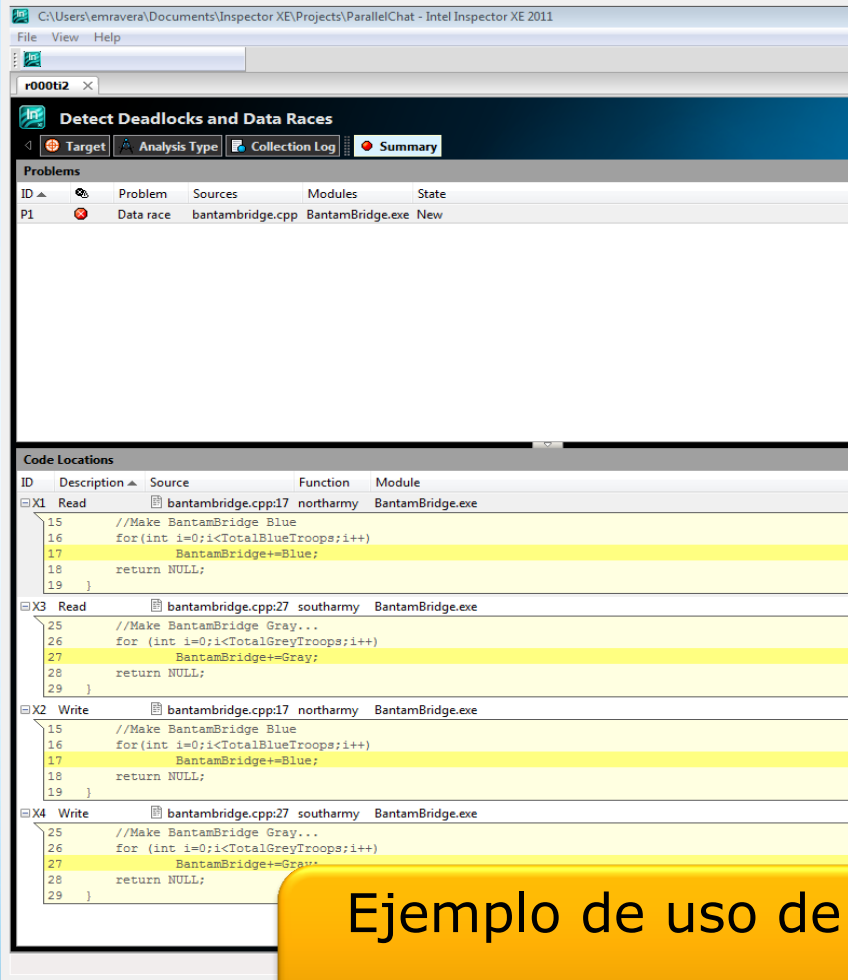
Una **race condition** es una conducta no-determinística causada por dos o más procesos accediendo en distintos momentos a una variable compartida - uno de los accesos modifica la variable.

Ejemplo :



Identificando Condiciones de Carrera

Usando  Intel® Inspector XE



The screenshot shows the Intel Parallel Inspector XE interface. The top menu bar includes File, View, and Help. Below it, the 'Detect Deadlocks and Data Races' section is active, with tabs for Target, Analysis Type, Collection Log, and Summary. The 'Problems' table lists a data race problem (P1) in bantambridge.cpp, BantamBridge.exe, with a state of 'New'. The 'Code Locations' table shows four locations (X1, X3, X2, X4) where the data race occurs, involving reads and writes to BantamBridge variables. The code snippets for these locations are displayed below the table.

ID	Description	Source	Function	Module
X1	Read	bantambridge.cpp:17	northarmy	BantamBridge.exe
X3	Read	bantambridge.cpp:27	southarmy	BantamBridge.exe
X2	Write	bantambridge.cpp:17	northarmy	BantamBridge.exe
X4	Write	bantambridge.cpp:27	southarmy	BantamBridge.exe

```
15 //Make BantamBridge Blue
16 for (int i=0;i<TotalBlueTroops;i++)
17     BantamBridge+=Blue;
18 return NULL;
19 }

25 //Make BantamBridge Gray...
26 for (int i=0;i<TotalGreyTroops;i++)
27     BantamBridge+=Gray;
28 return NULL;
29 }
```



Suppressed	
Not suppressed	1 item(s)
Investigated	
Not investigated	1 item(s)

Ejemplo de uso de Intel Parallel Inspector
Bantam Bridge (en C++)

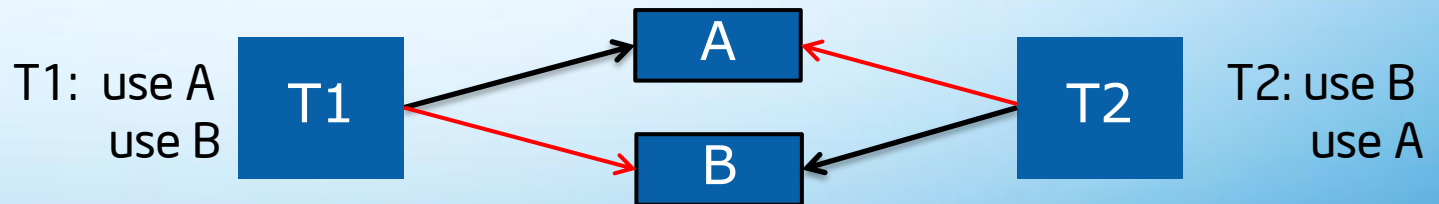
Abrazo Mortal

Deadlock se refiere a una condición donde dos o más procesos están esperando que el otro libere un recurso, en una cadena circular.

Condiciones Necesarias

1. Exclusión Mutua
2. Retención y Espera
3. No Expropiación
4. Espera Circular

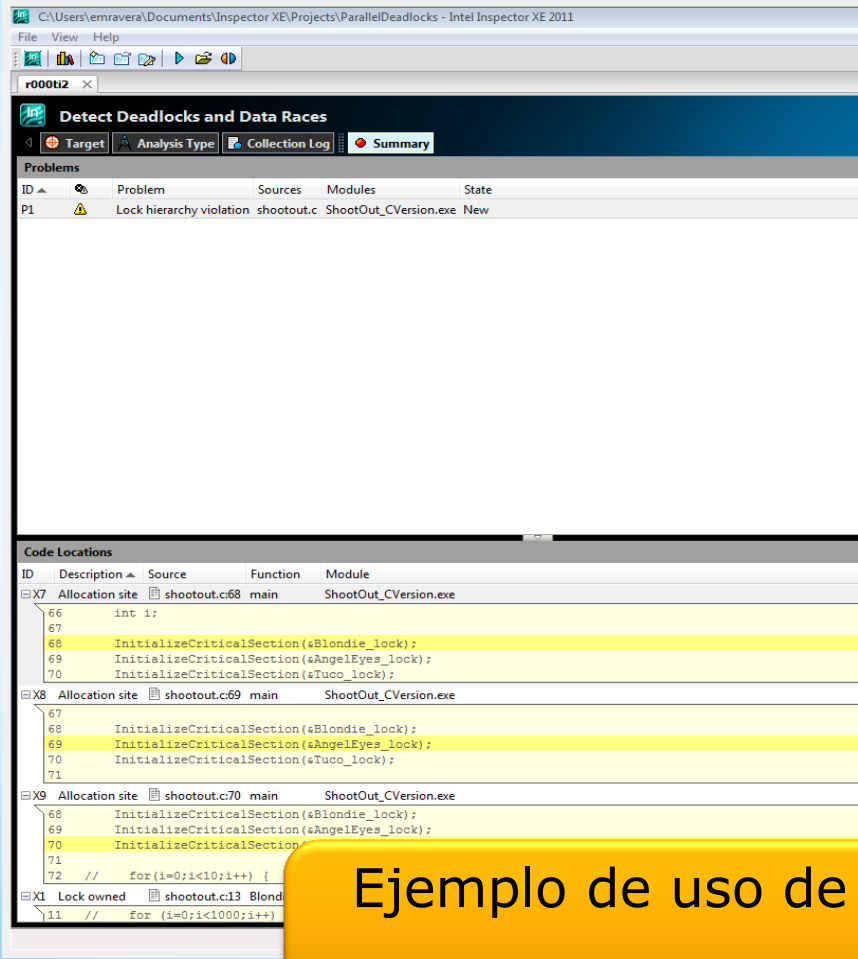
Ejemplo:



Ambos hilos (T1 y T2) están esperando que el otro libere el recurso

Detectando Deadlocks

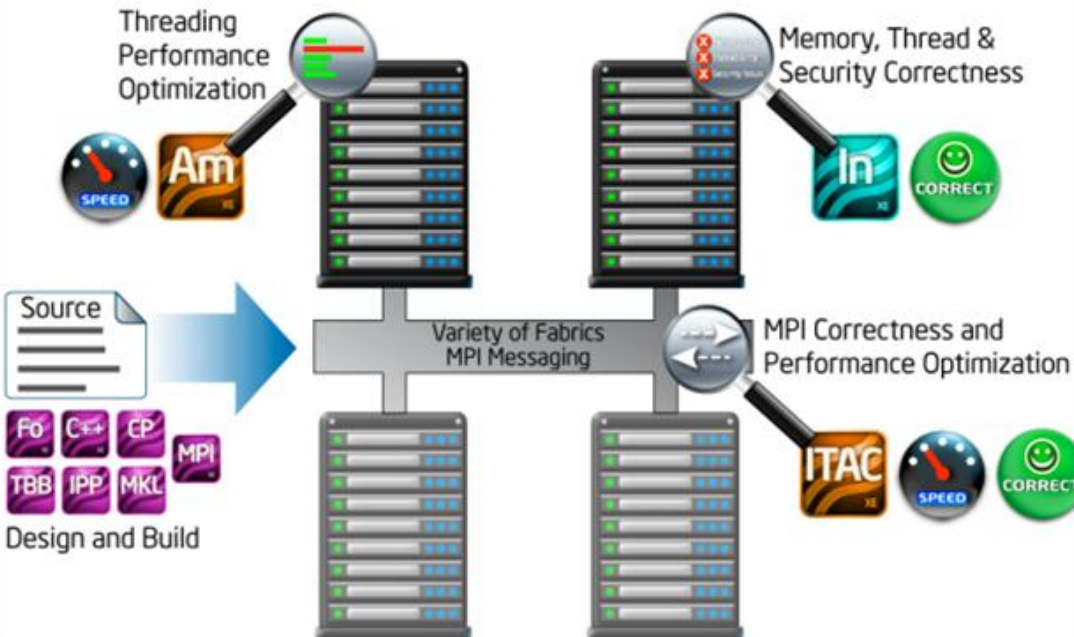
Usando  Intel® Inspector XE



Ejemplo de uso de Intel Parallel Inspector
Shoot Out (en C++)

Intel® Cluster Studio XE 2013

Intel® Cluster Studio XE *First comprehensive HPC hybrid parallelism development suite*



Herramienta de desarrollo basada en MPI para clústeres de alto desempeño (HPC)

Permite crear, analizar y optimizar aplicaciones para clústeres basadas en MPI.

Intel®
Cluster
Ready

Look for Intel® Cluster Ready solutions based on the latest Intel® Xeon® processors.



A modo de conclusión..

- Existen múltiples modelos de paralelismo en **hardware**, con los sistemas NUMA llevando la delantera.
- Existen múltiples modelos de paralelismo en **software**: multi-hilos & MPI
- Existen **herramientas** para ayudar en el desarrollo, verificación/validación y optimización de sistemas paralelos, debemos aprender a usarlas.
- No existe una **bala de plata**!

Intel Developer Zone

Soporte en todas las etapas de desarrollo

APRENDE



DESARROLLA



DISTRIBUYE



- Contenido técnico
- Tutoriales
- Trials gratuitos
- Foros y blogs
- Eventos
- Videos
- Tendencias
- Intel® Black Belt Software Developer

- Productos de desarrollo
- Cross-platform porting
- Acceso a Hardware
- Code Samples
- Librerías
- Compiladores
- Checkers
- Services APIs
- Testing
- Documentación
- Soporte de la comunidad

- Intel AppUp® Center
- Co-Marketing (promociones, eventos)
- Iniciativas Go-to-market
- Credencial Intel® Software Partner
- Collateral Builder



Ricardo Medel
ricardo.medel@intel.com