Práctica 4. Implementación, programación paralela y evaluación de prestaciones de multiprocesadores Nios II

Arquitectura de Computadores

2º GII (EII, ULPGC)

Objetivos

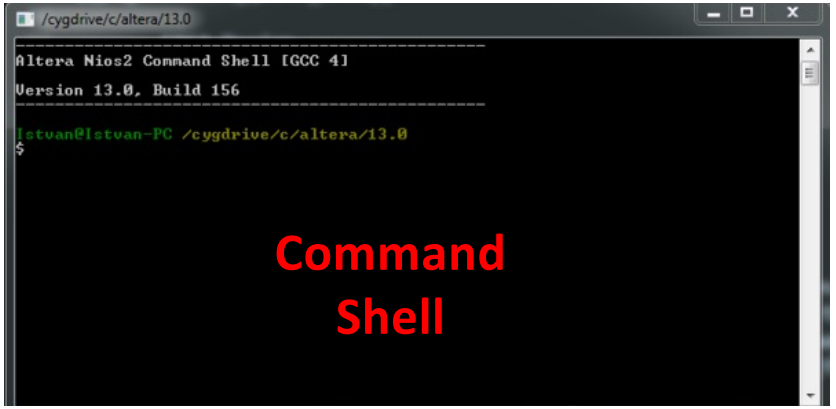
- Implementación de programas paralelos en dos multiprocesadores Nios II con el propósito de reducir significativamente el tiempo de ejecución de los programas.
- Registrar y comparar los tiempos de ejecución de los programas paralelos respecto a las correspondientes versiones secuenciales utilizando la placa DE0-Nano.
- Evaluar las prestaciones de los multiprocesadores Nios II para distintos volúmenes de datos procesados.
- Comparar las prestaciones de los distintos multiprocesadores Nios II.
- Implementar los programas paralelos en la placa DE0-Nano.
- Palabras clave: programación multihilos, paralelismo, multiprocesadores, sincronización de hilos, evaluación de prestaciones, Nios II, DE0-Nano.

Programación académica: 3 semanas

- Sesión 1:
  - Microarquitectura del computador paralelo Nios II + organización de la memoria principal + Entorno Software: SBT + Command Shell (1 h). Parte 1 del guion.
  - Tutorial 1: Hola_chicos_y_chicas (0,5 h). Parte 2 del guion.
  - Tutorial 2: Hola_semaforo (0,5 h). Parte 2 del guion.
- Sesión 2:
  - Tutorial 3: Matriz × Vector. Parte 3 del guion, Objetivos 3-1 y 3-2.
  - Propuesta ejercicio Matriz × Matriz. Objetivo 3-3.
- Sesión 3:
  - Realización del ejercicio Matriz × Matriz. Objetivo 3-3.
- Opcional (Objetivo 3-4)
  - Implementar Matriz × Vector y Matriz × Matriz usando el multiprocesador Nios II/f

Sesión 1

- Microarquitectura del computador paralelo Nios II
- Organización de la memoria principal
- Entorno Software:
  - Software Build Tool (SBT) para Eclipse
  - Command Shell de Windows
  - Máquina virtual Windows 7 en Virtual Box
- Tutorial 1: Hola_chicos_y_chicas
- Tutorial 2: Hola_semaforo



Introducción

- Hardware
  - Placas Intel/Altera DE0-Nano
  - Diseño FPGAs: Intel/Altera Quartus II + Qsys (ver: 12.1sp1 32-bit)
- Software
  - Nios II Embedded Design Suite (EDS) (ver: 12.1sp1 32-bit)
    - Software Build Tools (SBT) para Eclipse
    - Command Shell de Windows
  - Windows (7) en Virtual Box

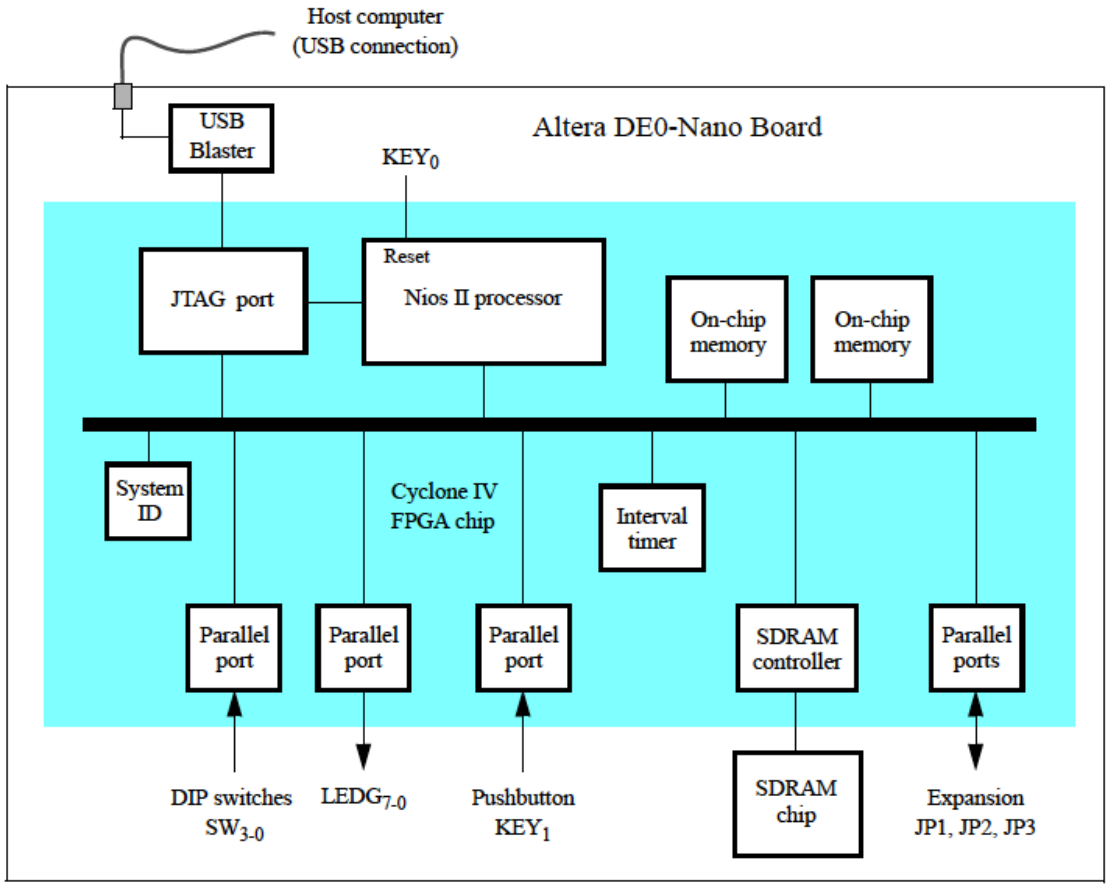
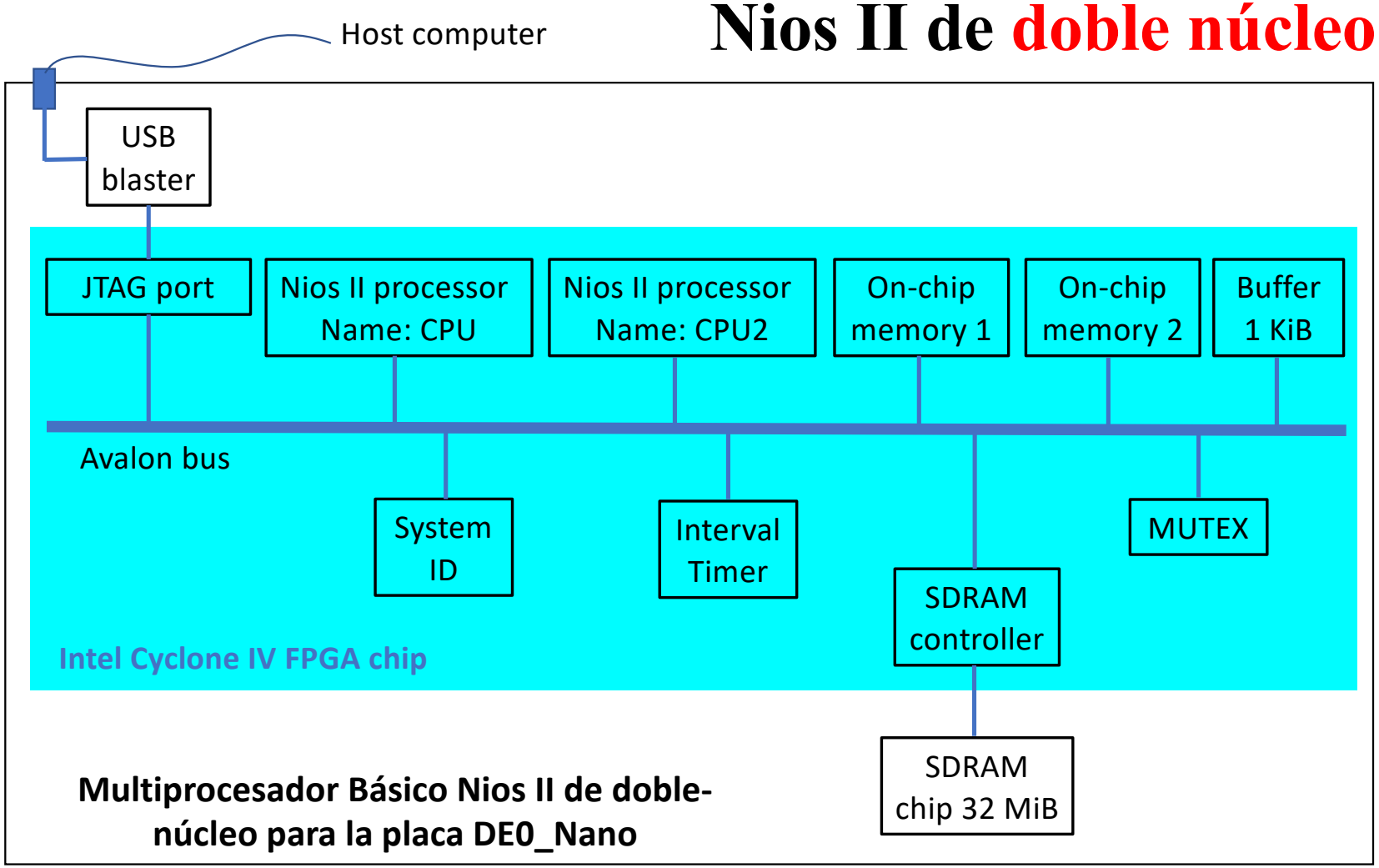


Figure 1. Block diagram of the DE0-Nano Basic Computer.

Introducción: Computador Básico Nios II de 1-núcleo (Prácticas 1, 2, 3)

- Computador Básico DE0-Nano
- Procesador:
  - Nios II/e
  - Arquitectura ISA de 32 bits
- Memoria SDRAM
  - DE0-Nano: 32 MiB SDRAM
- Memoria on-chip SRAM
  - DE0-Nano: 2 x 8 KiB

Introducción: Multiprocesador Nios II de doble núcleo (Práctica 4)

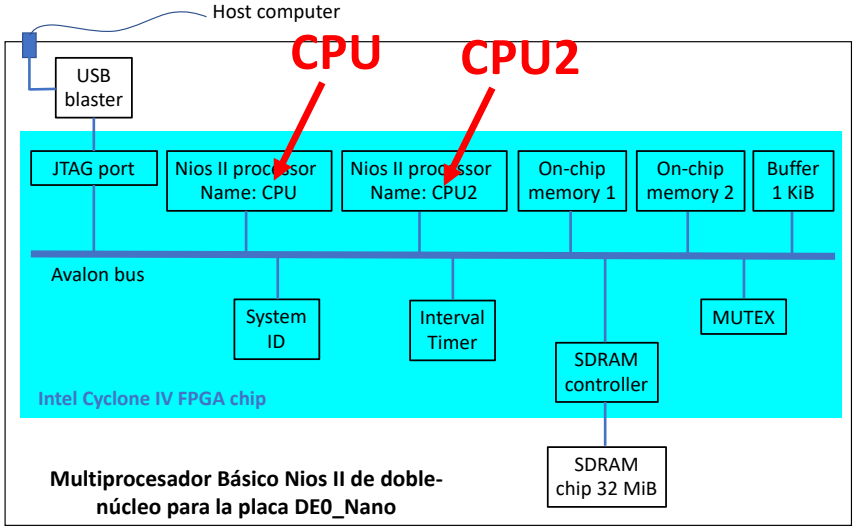


Multiprocesador Básico Nios II de doble-núcleo para la placa DE0_Nano

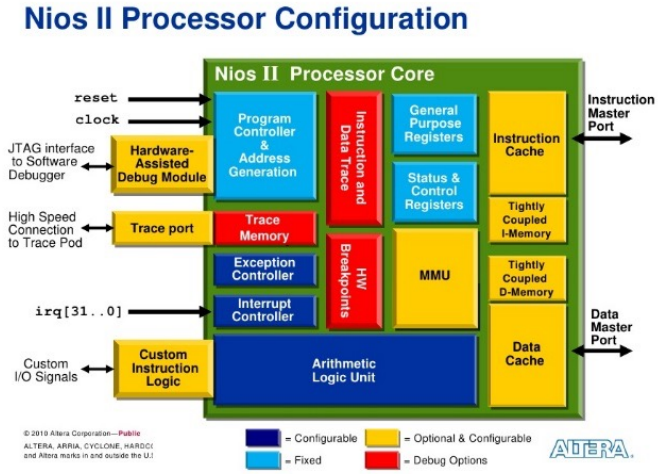
- 2 procesadores Nios II/{e,s}
- 1 semáforo, 8 bytes (MUTEX)
- 1 memoria para variables del semáforo, 1 KiB (BUFFER)

Implementación del multiprocesador de doble núcleo Nios II en la placa DE0-Nano

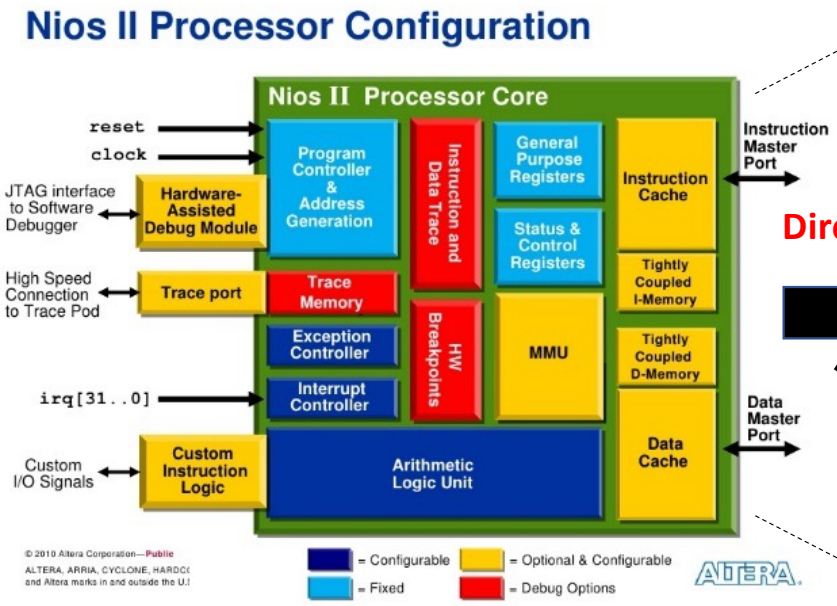
- Diseño FPGAs: Intel/Altera Quartus II + Qsys (ver: 12.1sp1 32-bit)
- Tipos de Multiprocesadores Doble-Núcleo Nios II (núcleos: **CPU** y **CPU2**)
  - 2 x Nios II/e (DualCoreNios2e): multiciclo, sin cache
  - 2 x Nios II/s (DualCoreNios2s): segmentado 5-etapas, con iCache (1KiB) + multiplicador hardware



Multiprocesador Básico Nios II de doble-núcleo para la placa DE0_Nano



Espacio de direccionamiento del procesador Nios II



Arquitectura de 32 bits

Dirección de 32 bits

Rangos de direcciones

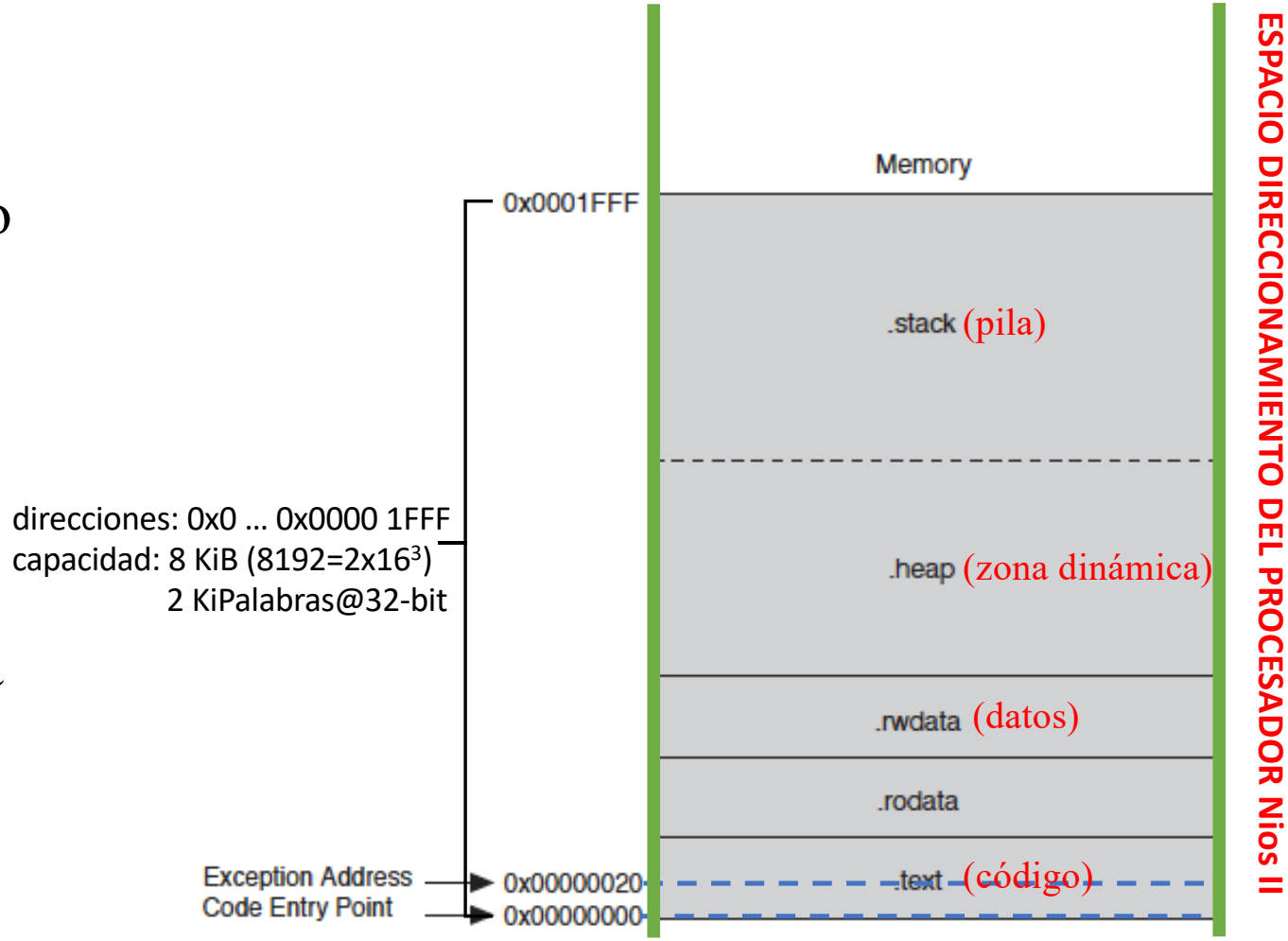
- Interval Timer CPU — 0x 1000 2000 - 0x 1000 201F (32 bytes)
- Interval Timer CPU2 — 0x 0B00 1000 - 0x 0B00 101F (32 bytes)
- on-chipMEM2/s1 — 0x 0900 0000 - 0x 0900 1FFF (8 KiB)
- on-chipMEM1/s1 — 0x 0800 0000 - 0x 0800 1FFF (8 KiB)
- BUFFER (message_buffer_ram/s1) — 0x 0400 0000 - 0x 0400 03FF (1 KiB)

MESSAGE_BUFFER_RAM_BASE

- MUTEX (message_buffer_mutex/s1) — 0x 0400 0400 - 0x 0400 0407 (8 bytes)
- SDRAM/s1 — 0x 0000 0000 - 0x 01FF FFFF (32 MiB)
- **CPU** — **0x 0000 0000 - 0x 003F FFFF (8 MiB)**
  - resetVector={SDRAM.s1,0x 0000 0000},
  - exceptionVector={SDRAM.s1,0x 0000 0020},
  - advanFeatures/cpuid=0x1
- **CPU2** — **0x 0040 0000 - 0x 007F FFFF (8 MiB)**
  - resetVector={SDRAM.s1,0x 0040 0000},
  - exceptionVector={SDRAM.s1,0x 0040 0020},
  - advanFeatures/cpuid=0x2

Ejemplo de organización de memoria para 1-core

- Punto de entrada del código
- Dirección de excepción
- Límite de memoria
- Capacidad de almacenamiento
- Rango de direcciones
- Particionado de la memoria en zonas de linkado:
  - código, datos, pila, zona dinámica

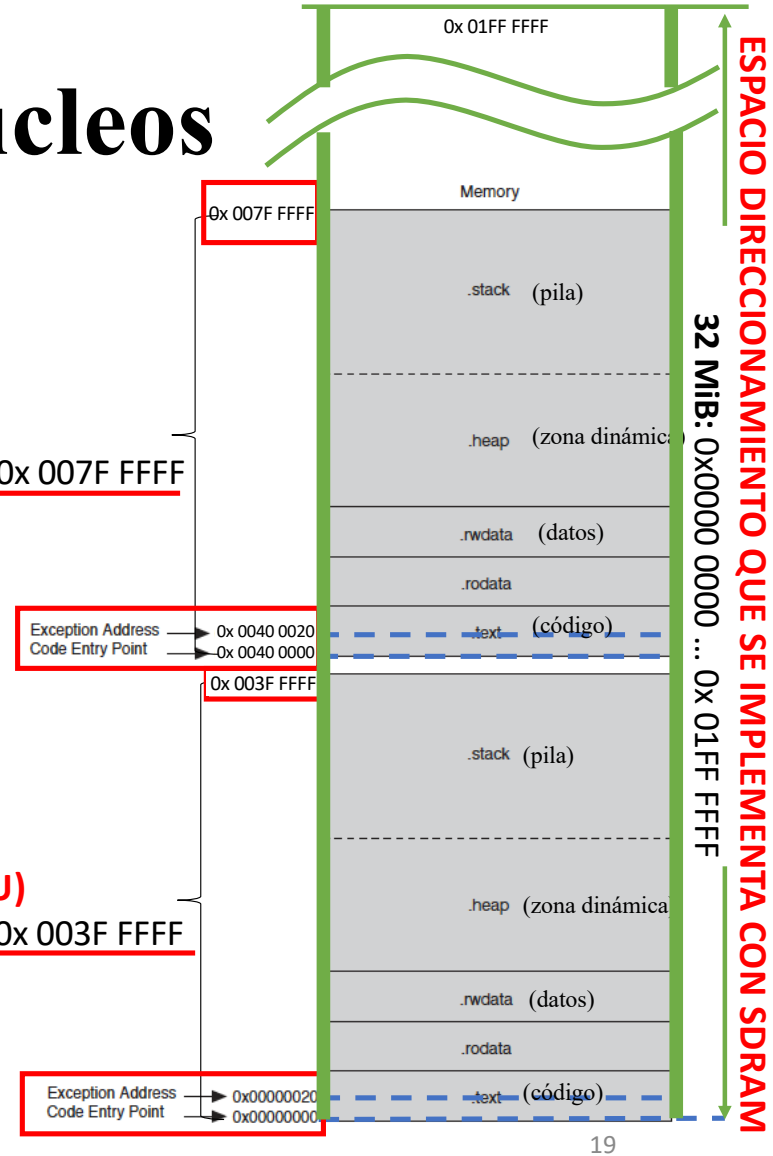


Organización de memoria 2-núcleos

- Puntos de entrada del código
- Direcciones de excepción
- Límites de memoria
- Capacidades de almacenamiento
- Rangos de direcciones
- Particionado de la memoria:
  - código, datos, pila, zona dinámica

Núcleo-2 (CPU2)
rango direcciones: 0x 0040 0000 … 0x 007F FFFF
capacidad: 4 MiB ($4x16^5$)

Núcleo-1 (CPU)
direcciones: 0x0000 0000 … 0x 003F FFFF
capacidad: 4 MiB ($4x16^5$)



Descripción y manejo de herramientas para el diseño software de programas que se ejecutan con arquitecturas Nios II

- Software Build Tools (SBT) para Eclipse, tareas:
  - Crear proyectos C, se necesita: Makefile, <file>.c, directorio BSP (system.h)
  - Compilar & Linkar, se genera: <file>.elf
- Command Shell, comandos:
  - Configurar placa DE0-Nano: $ nios2-configure <file>.sof (file DE0-nano= DE0_Nano_DualCore_Nios2e.sof)
  - Cargar programas en memoria y empezar ejecución en NiosII:
    `$ nios2-download -r -g -i <ID processor> <file>.elf`
  - Monitorizar resultados: `$ nios2_terminal.exe`
  - Tomar medidas de prestaciones: boli + papel
- Tutoriales para dos núcleos Nios II en esta práctica:
  - Tutorial 1 (secuencial): hola_chicos_y_chicas
  - Tutorial 2 (paralelo): hola_semaforo
  - Tutorial 3 (secuencial y paralelo): MV (Matriz x Vector)

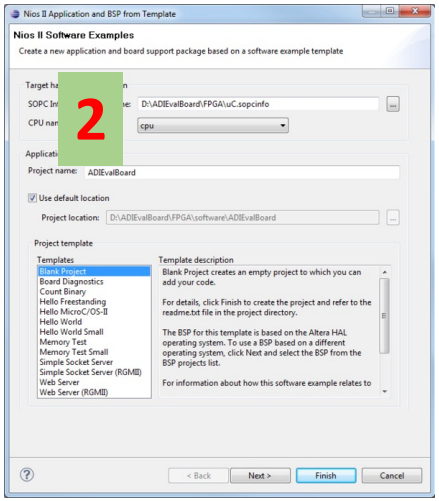
Tutorial-1 (1 núcleo): hola_chicos_y_chicas.c

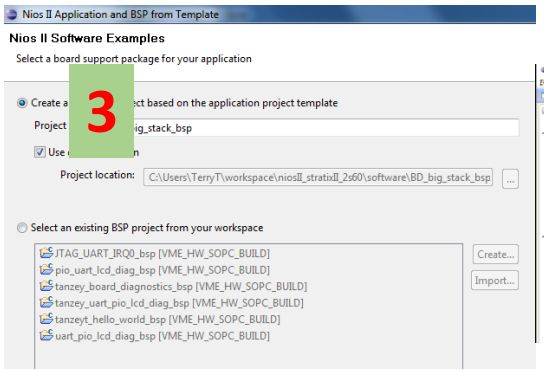
```
#include <stdio.h>

int main()
{
  printf("Hola chicos de AC!\n");
  printf("Hola chicas de AC!\n");

  return 0;
}
```

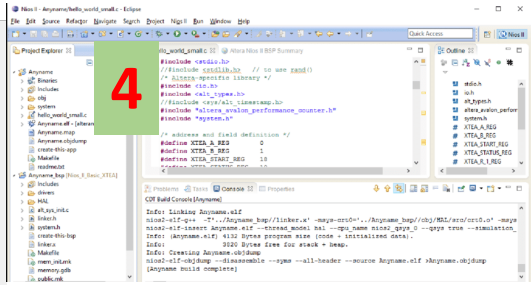
Tutorial-1 (1 núcleo): hola_chicos_y_chicas

1. Abrir: Nios II – Software Build Tools - Eclipse
2. File > New > Nios II app & BSP from Template
   a) SOPC Information File name: `DE0_Nano_DualCoreNios2e.sopcinfo`
   b) CPU name: **CPU** (también aparece CPU2 pero no se selecciona)
   c) Project name: hola_chicos_y_chicas_0
   d) Project Location: elegir <ProjectDir>
   e) Templates: seleccionar “Hello World Small”
   f) Picar en “Next >”
3. Seleccionar “Create BSP”, Project name: hola_chicos_y_chicas_0.bsp, Use default location, picar en “Finish”
   a) Resultado: en la ventana Eclipse “Project Explorer” se crea un proyecto de tipo “app C/C++”, y otro proyecto de tipo “BSP”
4. Desplegar el proyecto hola_chicos_y_chicas_0 en la ventana “Project Explorer” y picar 2 veces en hello_world.c
   a) Resultado: se abre una ventana con el código fuente.
5. Sustituir el fichero hello_world.c del directorio del proyecto por el fichero Tutorial1/hola_chicos_y_chicas.c
6. Compilar y linkar
   a) Picar botón derecho sobre proyecto C/C++ en ventana Project Explorer
   b) Seleccionar: Build Project
   c) Resultado: si el proceso termina bien, aparece en la ventana inferior “Console” el mensaje: "hola_chicos_y_chicas_0 Build complete”. Se crean dos proyectos en ventana Project Explorer y dos carpetas en <ProjectDir>

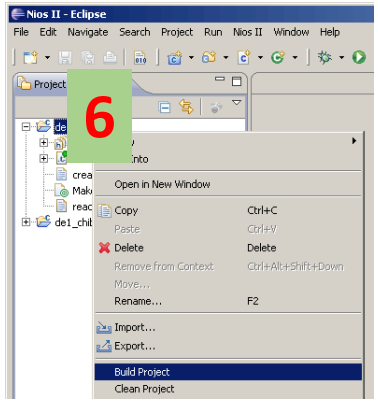
Tutorial-1 (1 núcleo): hola_chicos_y_chicas





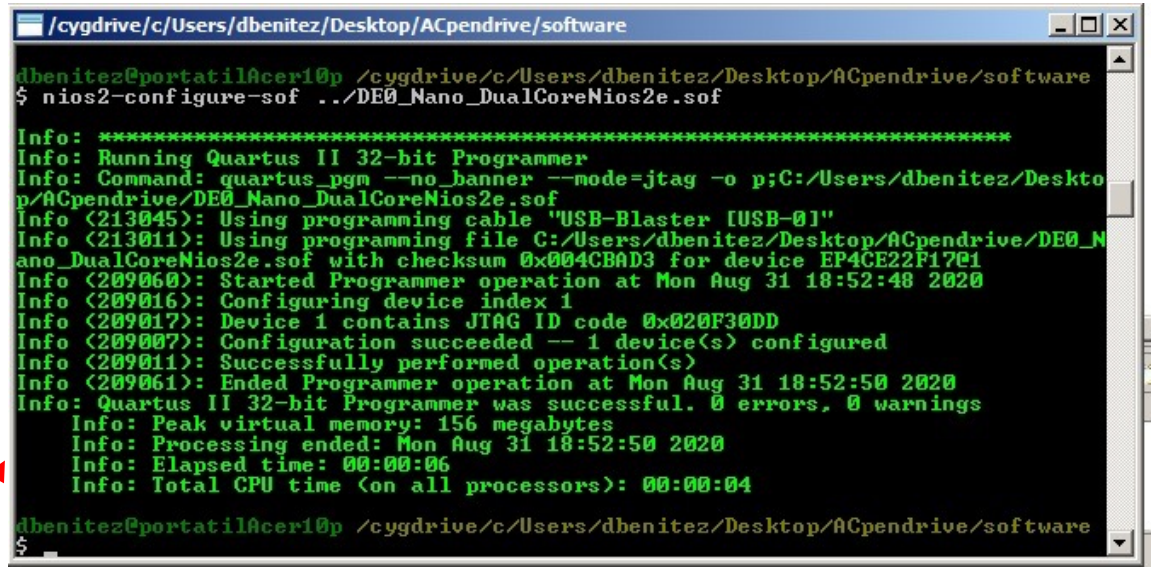


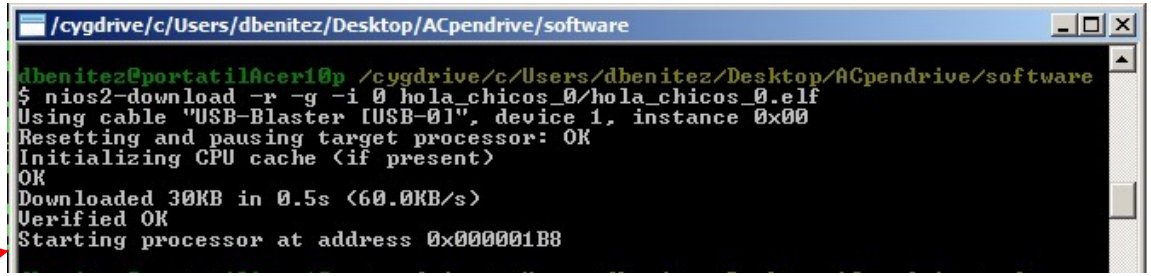
- Pasos a realizar en el Tutorial utilizando el entorno Eclipse

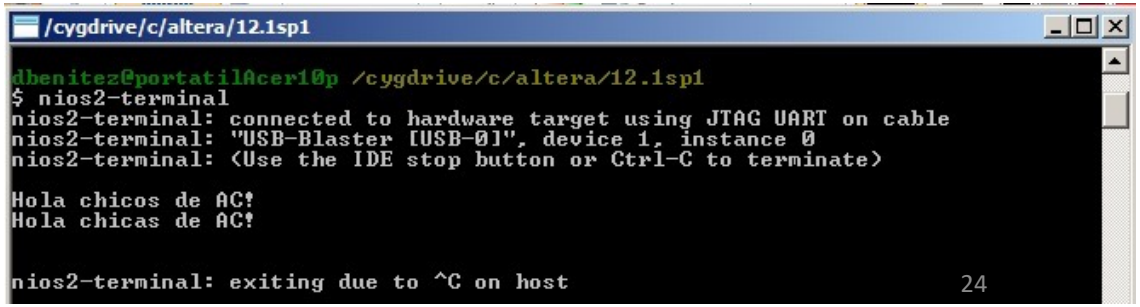


Tutorial-1 (1 núcleo): hola_chicos_y_chicas

7. Configuración placa DE0-Nano:
   a) Conectar placa a conector USB (ya está conectada en el Laboratorio)
   b) Abrir: Inicio > Altera > NiosII Command Shell
   c) Se carga fichero DE0_Nano_Basic_Computer.sof en placa :
      1. `$ cd …/Users/…/Desktop/…/configuracionesFPGA`
      2. `$ nios2-configure-sof DE0_Nano_DualCoreNios2e.sof`
      (intentarlo varias veces si da error de JTAG)
   d) Resultado: tiene que aparecer el mensaje "Quartus II 32-bit Programmer was successful"
8. Ejecución del programa:
   a) `$ cd <proyecto hola_chicos_y_chicas_0>`
   b) `$ nios2-download -r -g -i 0 hola_chicos_y_chicas_0.elf`
9. Visualizar resultados:
   a) `$ nios2-terminal`
      - Resultado: aparecerán los mensajes en la ventana correspondientes a los printf del código fuente
   b) Cerrar ventana de Command Shell







Tutorial-2: programación multihilos

- La metodología de programación de los multiprocesadores de doble núcleo que usaremos en esta práctica se denomina *programación multihilos*.
- Consiste en la realización de dos programas que se sincronizan usando
  - variables compartidas en el espacio de direccionamiento de los procesadores
  - además de un mecanismo hardware de exclusión mutua denominado *semáforo*.

Tutorial-2: programación multihilos
2 núcleos: CPU, CPU2 2 hilos: hola_semaforo_{0,1}.c

CPU (núcleo 1)

```
/* Tutorial: "hola_semaforo_0.c" */
#include <stdio.h>
#include <system.h>
#include <altera_avalon_mutex.h>
#include <unistd.h>

int main(){

// dirección de memoria de message buffer: 0x 0400 0000
volatile int * message_buffer_ptr = (int *) MESSAGE_BUFFER_RAM_BASE;

printf("Hola, soy CPU!\n");

/* se guarda el manejador del dispositivo hardware de tipo mutex */
alt_mutex_dev* mutex = altera_avalon_mutex_open("/dev/message_buffer_mutex");

int message_buffer_val   = 0x0;
int iteraciones          = 0x0;

while(1) {
  iteraciones++;
  /* CPU pide ser propietario de mutex, asignando el valor 1 */
  altera_avalon_mutex_lock(mutex,1);
  message_buffer_val = *(message_buffer_ptr); /* lee valor guardado en buffer */
  altera_avalon_mutex_unlock(mutex); /* libera mutex */
  printf("CPU - iter: %i - message_buffer_val: %08X\n", iteraciones, message_buffer_val);
  usleep(1000000); /* espera 1 seg = 10^6 useg */
  }
return 0;
}
```

CPU LEE la variable message_buffer_val

CPU2 (núcleo 2)

```
/* Tutorial: "hola_semaforo_1.c" */
#include <stdio.h>
#include <system.h>
#include <altera_avalon_mutex.h>

int main(){

// dirección de memoria de message buffer: 0x 400 0000
volatile int * message_buffer_ptr  = (int *)
                                MESSAGE_BUFFER_RAM_BASE;

/* se guarda el manejador del dispositivo hardware de tipo mutex */
alt_mutex_dev* mutex =
altera_avalon_mutex_open("/dev/message_buffer_mutex");

int message_buffer_val    = 0x0;

while(1) {
  /* CPU pide ser propietario de mutex, asignando el valor 2 */
  altera_avalon_mutex_lock(mutex,2);
   /* guarda en buffer el valor modificado en CPU2 */
  *(message_buffer_ptr) = message_buffer_val;
  altera_avalon_mutex_unlock(mutex); /* libera mutex */
  message_buffer_val++;
  }
return 0;
}
```

Sólo CPU2 MODIFICA la variable message_buffer_val

Aclaración

- El espacio de direccionamiento de los procesadores es de 32 bits: 4 GiB (0x0000 0000 - 0xFFFF FFFF)

- En este espacio están mapeados distintos dispositivos hardware asignándoles distintos rangos de direcciones, entre ellos:
- dispositivo message_buffer_ram: 0x 0400 0000 - 0x 0400 03FF (1 KiB)
- dispositivo on-chip SRAM 1: 0x 0800 0000 - 0x 0800 1FFF (8 KiB)
  dispositivo on-chip SRAM 2: 0x 0900 0000 - 0x 0900 1FFF (8 KiB)
  dispositivo SDRAM: 0x 0000 0000 - 0x 01FF FFFF (32 MiB)

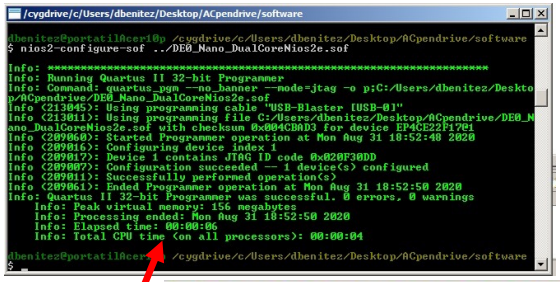
- La constante MESSAGE_BUFFER_RAM_BASE está igualada a la dirección 0x 0400 0000 que se encuentra asignada al dispositivo message_buffer_ram y no al dispositivo SDRAM. Cuando se accede a MESSAGE_BUFFER_RAM_BASE, no se está accediendo a la SDRAM sino al dispositivo message_buffer_ram.

Tutorial-2 (2 núcleos): hola_semaforo

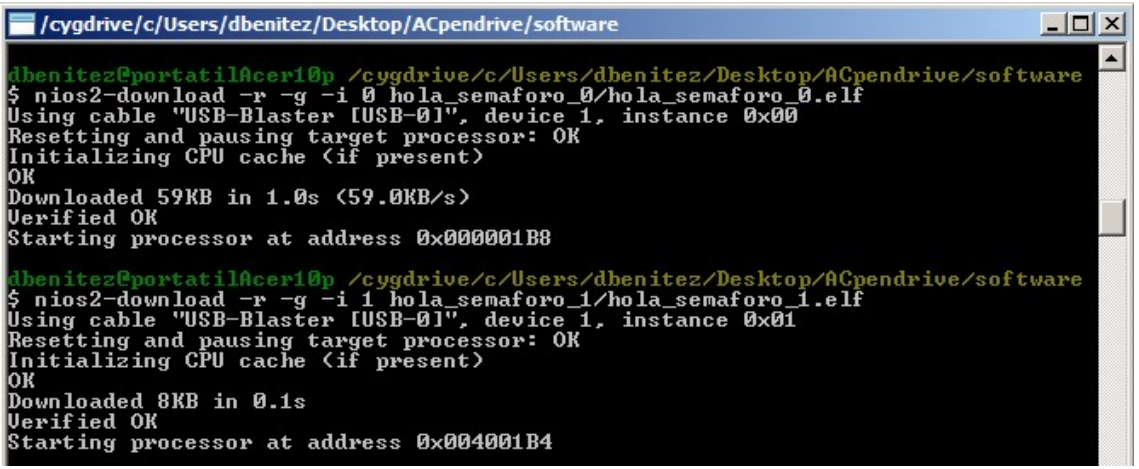
Se crean dos proyectos siguiendo los pasos del Tutorial-1

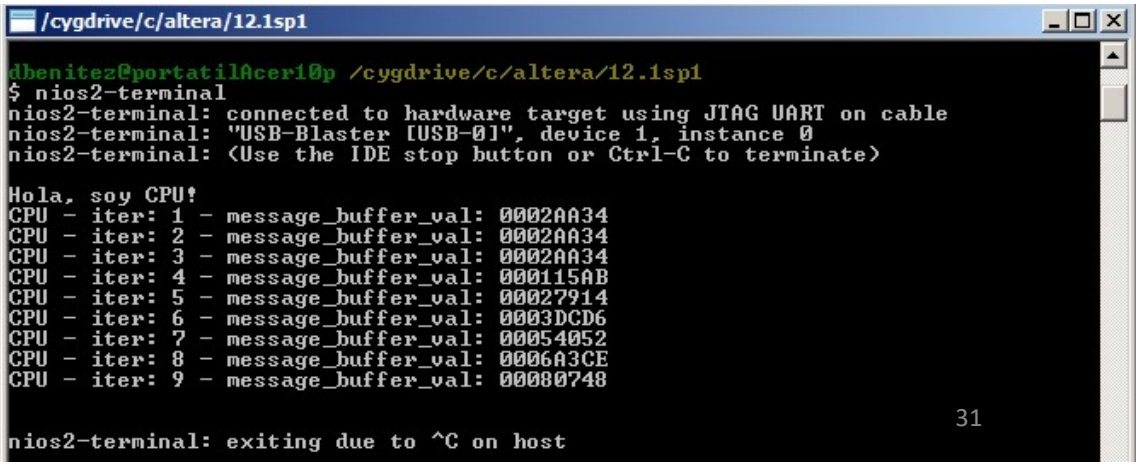
1. Elementos comunes de ambos proyectos:
    1. SOPC Information File name: DE0_Nano_DualCoreNios2e.sopcinfo
    2. Project Location: elegir <ProjectDir>
    3. Templates: seleccionar “Hello World Small”
2. Proyecto 1:
    1. CPU name: CPU (también aparece CPU2 pero no se selecciona)
    2. Project name: hola_semaforo_0
    3. Poyecto BSP, Project name: hola_chicos_y_chicas_0.bsp
    4. Sustituir el fichero hello_world.c del directorio del proyecto por el fichero Tutorial2/hola_semaforo_0.c
    5. Compilar y linkar
3. Proyecto 2:
    1. CPU name: CPU2
    2. Project name: hola_semaforo_1
    3. Poyecto BSP, Project name: hola_chicos_y_chicas_1.bsp
    4. Sustituir el fichero hello_world.c del directorio del proyecto por el fichero Tutorial2/hola_semaforo_1.c
    5. Compilar y linkar

Tutorial-2 (2 cores): hola_semaforo



4. Configuración placa DE0-Nano:
   - Conectar placa a conector USB (ya está conectada en el Laboratorio)
   - Abrir: Inicio > Altera > NiosII Command Shell
   - Se carga fichero DE0_Nano_Basic_Computer.sof en placa :
     - ❑ $ cd …/Users/…/Desktop/…/configuracionesFPGA
     - ❑ $ nios2-configure-sof DE0_Nano_DualCoreNios2e.sof (intentarlo varias veces si da error de JTAG)
   - Resultado: tiene que aparecer el mensaje "Quartus II 32-bit Programmer was successful"
5. Ejecución del programa:
   - $ nios2-download -r -g **-i 0** hola_semaforo_0/hola_semaforo_0.elf
   - $ nios2-download -r -g **-i 1** hola_semaforo_1/hola_semaforo_1.elf
6. Visualizar resultados:
   - $ nios2-terminal
   - $ CTRL-C





Tutorial 3: programación paralela multihilos y evaluación de prestaciones en un multiprocesador Nios II de doble núcleo

- Benchmark : multiplicación **Matriz x Vector**
- Objetivo 3-1: evaluación de prestaciones del benchmark secuencial
- Objetivo 3-2: evaluación de prestaciones del benchmark paralelo para 2-núcleos: 2 x Nios II/e y 2 x Nios II/s
- Objetivo 3-3: ejercicio de implementación del programa de multiplicación **Matriz x Matriz** en 2-núcleos Nios II y evaluar prestaciones

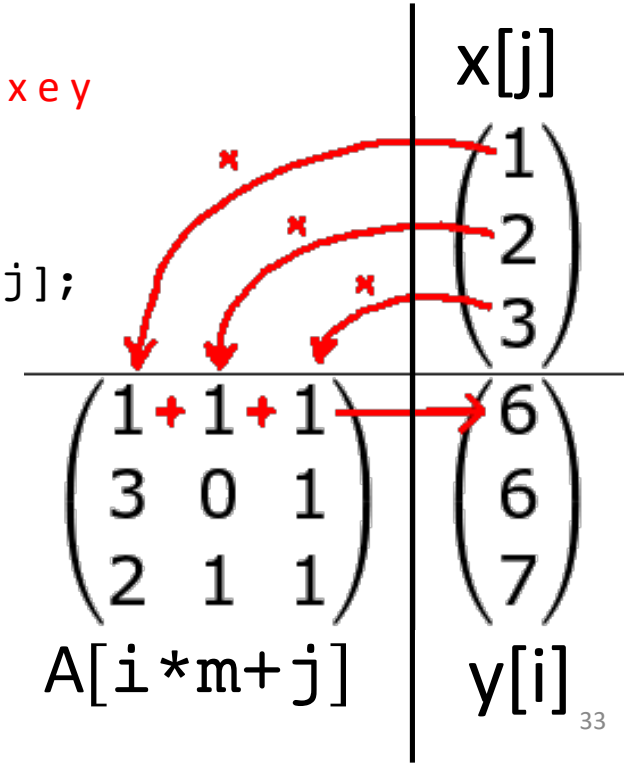
Benchmark: multiplicación Matriz x Vector

- La operación matemática es : $y = A \cdot x$
- Código principal en C:

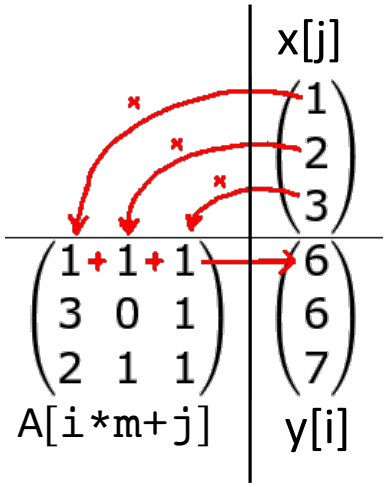
n : número de filas de la matriz y número de elementos de los vectores x e y

m : número de columnas

```
for (i=0; i<n; i++) {
  for(j=0; j<m; j++) y[i] += A[i*m+j] * x[j];
}
```



Benchmark: implementación Matriz x Vector



- Bucle de `Niter` iteraciones en el que cada iteración se multiplica una matriz: `n`(filas)=16 x `m`(columnas)=16 de valores enteros (**`A[i*m+j]`**) y un vector de 16 valores enteros (**`x[j]`**). El resultado consiste en 16 valores enteros (**`y[i]`**).
- Registro de tiempos con la función HAL: `alt_timestamp()`
- Visualización de resultados en CPU: `printf()`. Solo CPU está conectado de la interfaz JTAG por lo que su programa es el único que puede tener `printf`. Todo printf en CPU2 no se puede visualizar.

Benchmark: carga de trabajo Matriz x Vector en el programa secuencial

- Código principal en C:

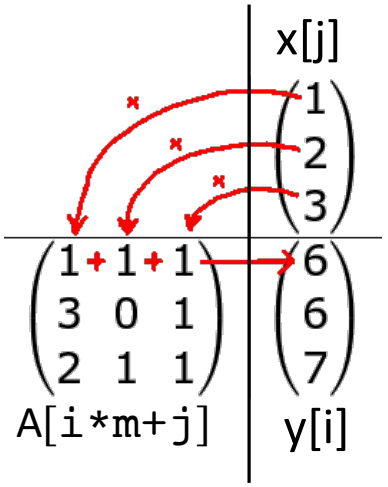
Núcleo 1: rank = 0

thread_count = 1

```
int local_n         = n / thread_count;
int my_first_row    = rank * local_n;          // 1™ fila asignada a cada hilo
int my_last_row     = (rank+1) * local_n - 1;  // ultima fila asignada a cada hilo

for (k1 = 0; k1 < Niter; k1++) {
        iteraciones++;
        for (i = my_first_row; i <= my_last_row; i++) {
            dumy = y[i];
            for(j = 0; j < m; j++){
                    dumy += A[i*m+j] * x[j];
            }
            y[i] = dumy;
        }
}
```

0

n-1

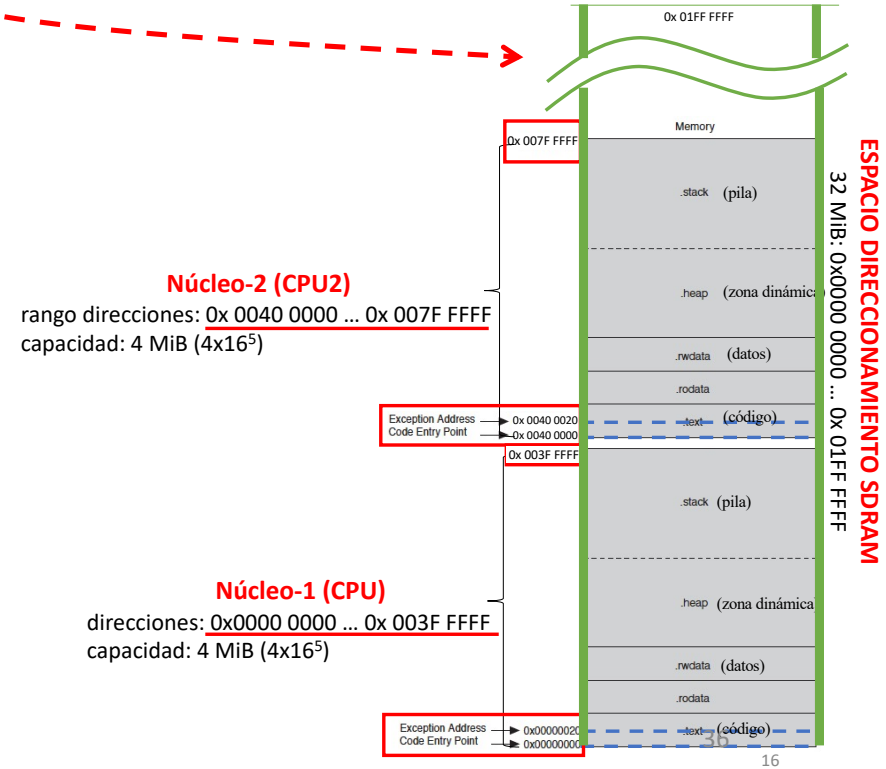


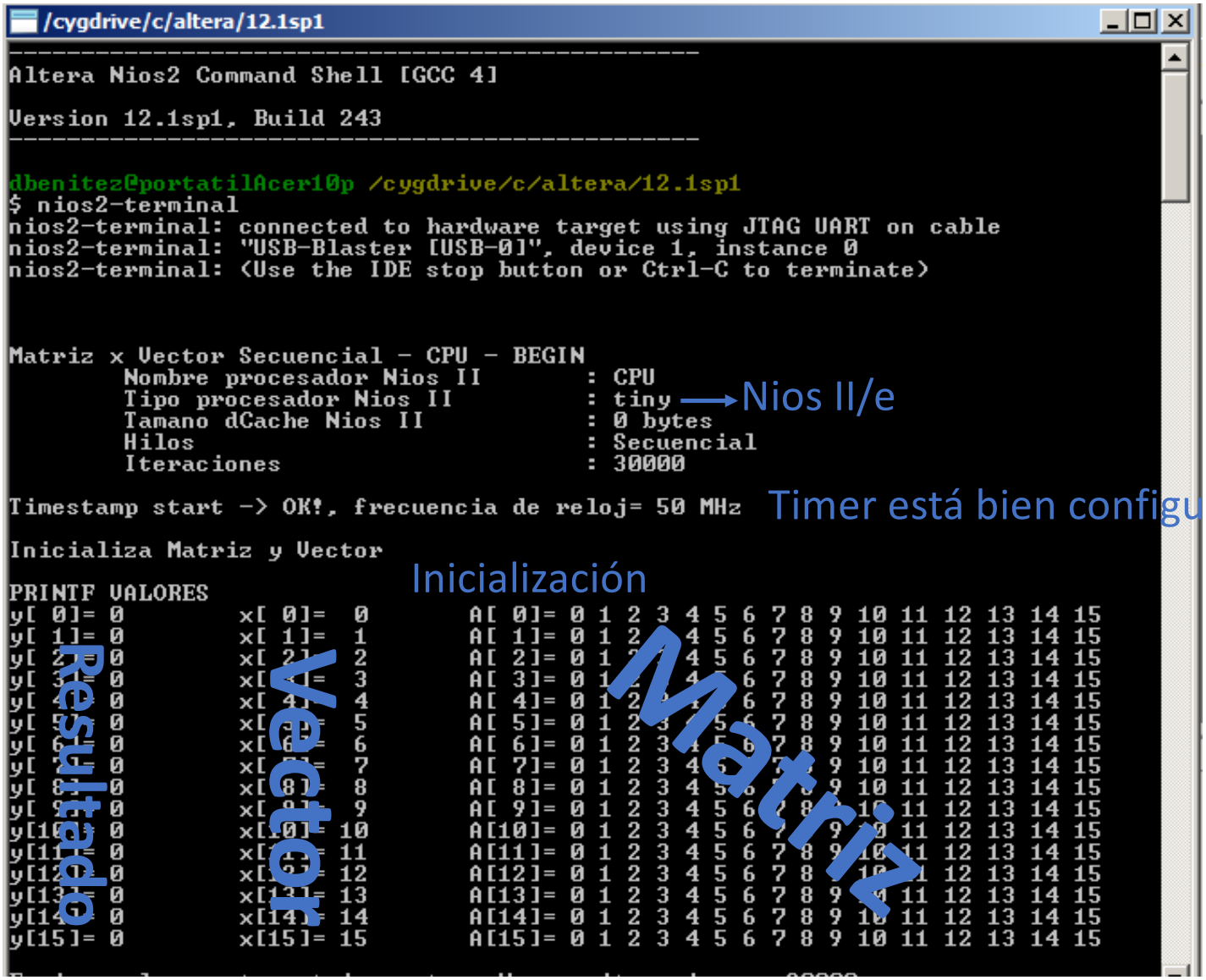
Benchmark secuencial para 1-núcleo: código fuente (MV_serie.c)

```
int main(){
// zona de memoria compartida para matriz y vectores
volatile int * A      = (int *) 0x806000;  // 16x16x4=1KiB: 0x806000 - 0x8063FF
volatile int * x      = (int *) 0x806400;  // 16x1  x4=64 B: 0x806440 - 0x80647F
volatile int * y      = (int *) 0x806800;  // 16x1  x4=64 B: 0x806480 - 0x8064BF

// CÓMPUTO - Operación Matriz x Vector
int local_n          = n;
int my_first_row = 0;                 // 1™ fila asignada a este nucleo
int my_last_row  = local_n - 1;  // ultima fila asignada a este nucleo
for (k = 0; k < Niter; k++) {
            iteraciones++;
            for (i = my_first_row; i <= my_last_row; i++) {
              for(j = 0; j < m; j++)    y[ i ] += A[ i*m + j ] * x[ j ];
            }
}
} // main()
```

Repeticiones del bucle Matriz-Vector

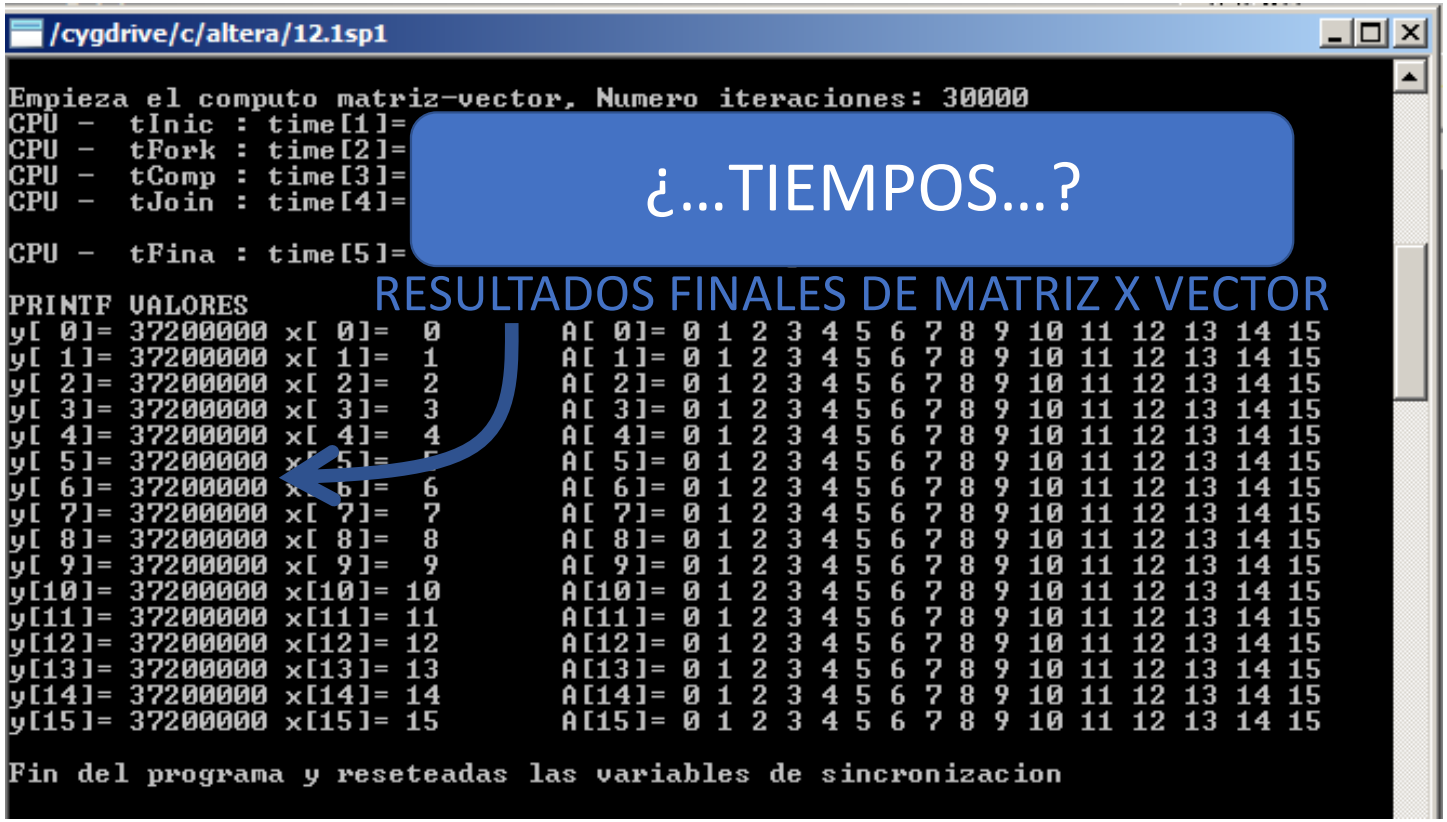
Asignación de carga computacional



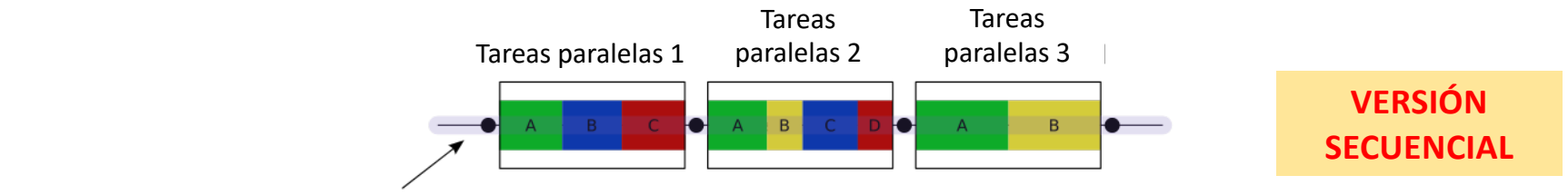


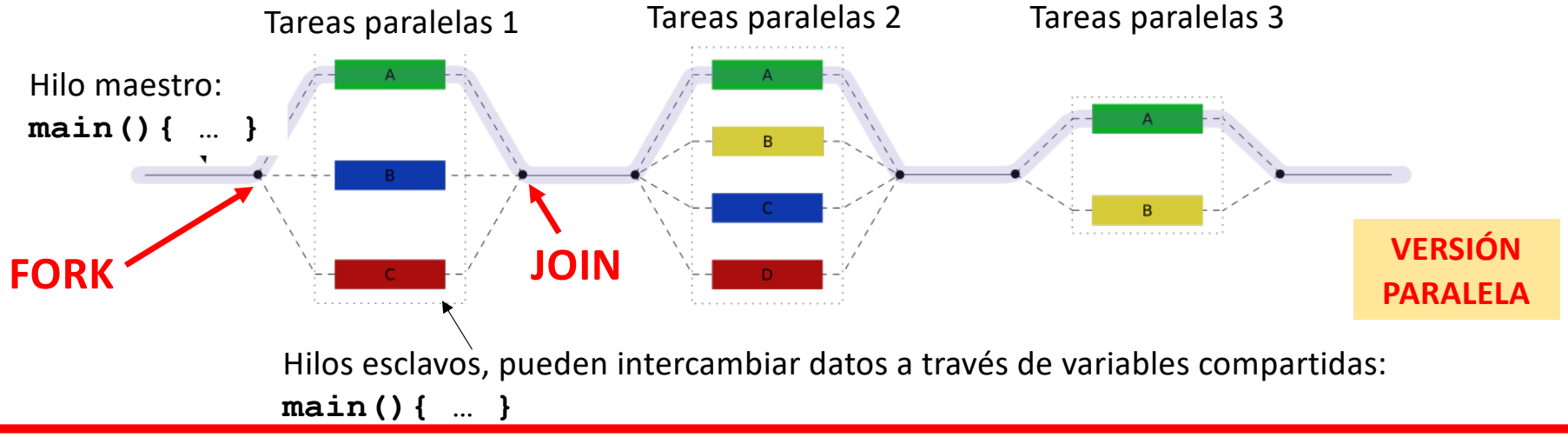
Ejecución del programa secuencial en uno de los núcleos del multiprocesador DualCoreNios2e

Ejecución del programa secuencial en uno de los núcleos del multiprocesador DualCoreNios2e

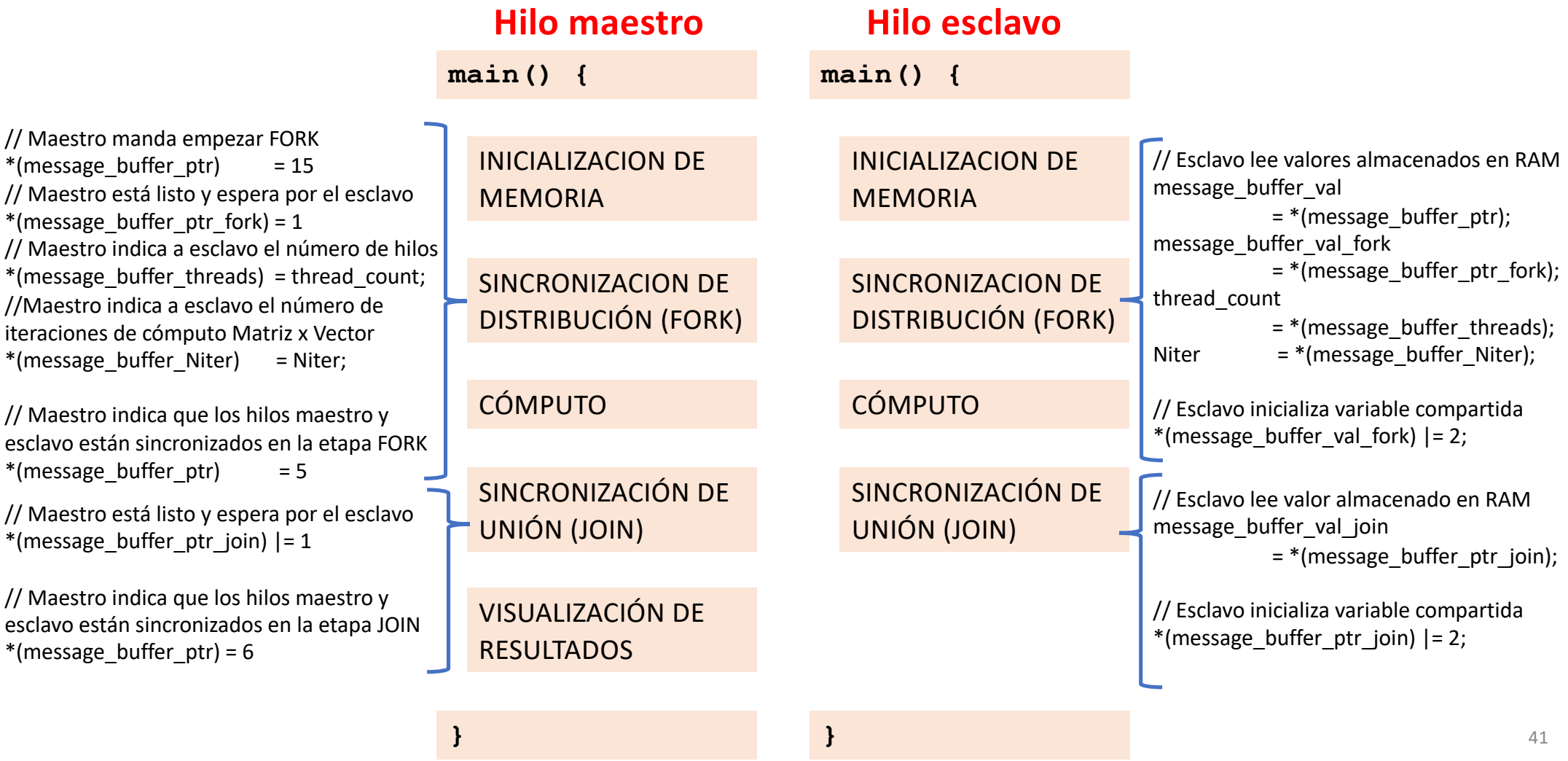


Modelo Fork/Join de programación paralela





Programa paralelo multihilo



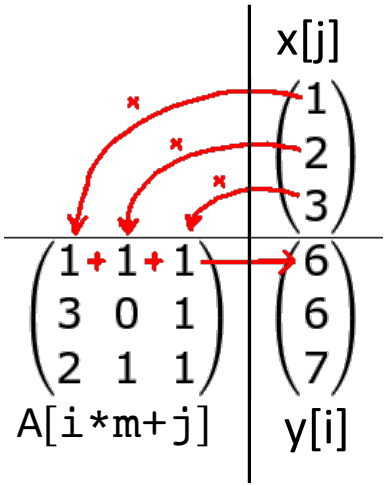
Benchmark: partición de la carga de trabajo Matriz x Vector en el programa paralelo

CÓMPUTO

- Código principal en C de los dos hilos/programas:

Núcleo/Hilo 1: rank = 0
Núcleo/Hilo 2: rank = 1
thread_count = 2
Niter = …

```
int local_n         = n / thread_count;
int my_first_row    = rank * local_n;          // 1™ fila asignada a cada hilo
int my_last_row     = (rank+1) * local_n - 1;  // ultima fila asignada a cada hilo

for (k1 = 0; k1 < Niter; k1++) {
        iteraciones++;
        for (i = my_first_row; i <= my_last_row; i++) {
            dumy = y[i];
            for(j = 0; j < m; j++){
                    dumy += A[i*m+j] * x[j];
            }
            y[i] = dumy;
        }
}
```



Benchmark para multiprocesador 2-núcleos: código fuente, Parte 1 (MV_paralelo_maestro.c)

CPU (núcleo 1)

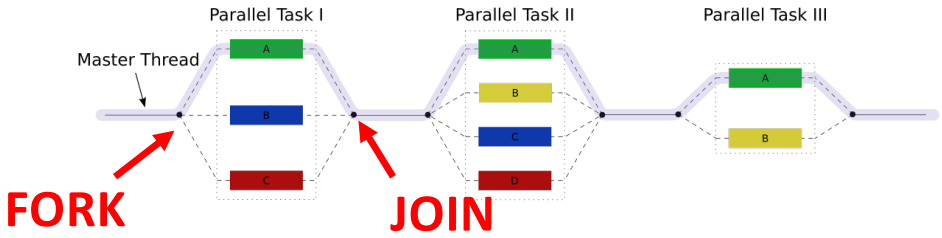
```
// RAM para sincronizacion entre hilos, tamano total RAM= 20 bytes
// Todas las variables se encuentran en una linea de cache (32 bytes)
volatile unsigned int * message_buffer_ptr          = (unsigned int *) MESSAGE_BUFFER_RAM_BASE;
volatile unsigned int * message_buffer_ptr_join     = (unsigned int *) (MESSAGE_BUFFER_RAM_BASE+4);
volatile unsigned int * message_buffer_ptr_fork     = (unsigned int *) (MESSAGE_BUFFER_RAM_BASE+8);
volatile unsigned int * message_buffer_threads      = (unsigned int *) (MESSAGE_BUFFER_RAM_BASE+12);
volatile unsigned int * message_buffer_Niter        = (unsigned int *) (MESSAGE_BUFFER_RAM_BASE+16);
// Zona de memoria compartida para matrices A,B,C
volatile int * A = (int *) 0x806000; // 16x16x4=1KiB: 0x806000 - 0x8063FF
volatile int * x = (int *) 0x806400; // 16x16x4=1KiB: 0x806400 - 0x8067FF
volatile int * y = (int *) 0x806800; // 16x16x4=1KiB: 0x806800 - 0x806BFF
#define m 16 // numero de columnas de las matrices
#define n 16 // numero de filas de las matrices
int main() {
…
}
```

SINCRONIZACION DE DISTRIBUCIÓN (FORK)

Benchmark para multiprocesador 2-núcleos: código fuente, Parte 2 (MV_paralelo_maestro.c)

CPU (núcleo 1)

```
int main() {
alt_mutex_dev* mutex = altera_avalon_mutex_open("/dev/message_buffer_mutex");
int thread_count = 2;       // numero de hilos
int rank         = 0;       // hilo maestro para nucleo= CPU
int Niter        = 2000; // veces repite matriz-vector; otros valores: 1000,2000,5000,10000
// FORK - Sincronizacion de Separacion: *MESSAGE_BUFFER_RAM_BASE = 15
        message_buffer_val          = 15;                    // ID=15(0xF) indica que Fork empieza
        message_buffer_val_fork     = 1 ;                    // indica que maestro esta preparado para computo
        altera_avalon_mutex_lock(mutex,1);                   // bloquea mutex
        *(message_buffer_ptr_dele)  = 0;                     // obliga a vaciar la línea de cache para obligar a leer variables desde RAM
        *(message_buffer_ptr)       = message_buffer_val;    // inicializa RAM GLOBAL
        *(message_buffer_ptr_fork)  = message_buffer_val_fork; // inicializa RAM FORK
        *(message_buffer_threads)   = thread_count;          // inicializa RAM THREADS
        *(message_buffer_Niter)     = Niter;                 // inicializa RAM NITER
        altera_avalon_mutex_unlock(mutex);                   // libera mutex
        while( (message_buffer_val != 5) ){          // bucle while de sincronización FORK; message_buffer_val = 5 : indica que hilos estan sincronizados para computo
                    … siguiente transparencia …
        }
```



SINCRONIZACION DE DISTRIBUCIÓN (FORK)

Benchmark para multiprocesador 2-núcleos: código fuente, Parte 3 (MV_paralelo_maestro.c)

```
int main() {
…
// FORK - Sincronizacion de Separacion: *MESSAGE_BUFFER_RAM_BASE = 15
…
        while( (message_buffer_val != 5) ){          // bucle while de sincronización FORK; message_buffer_val = 5 : indica que hilos estan sincronizados para computo
                altera_avalon_mutex_lock(mutex,1);                                  // bloquea mutex
                *(message_buffer_ptr_dele)  = 0;                                    // obliga a vaciar la cache
                message_buffer_val_fork     = *(message_buffer_ptr_fork);           // lee valor en RAM para ver si el otro procesador esta listo
                *(message_buffer_ptr_dele)  = 0;                                    // obliga a vaciar la cache
                altera_avalon_mutex_unlock(mutex);                                  // libera mutex
                if ( (message_buffer_val_fork == 0x3 && thread_count == 2 ) ||
                        (message_buffer_val_fork == 0x1 && thread_count == 1) ){
                        dumy = 5;
                        altera_avalon_mutex_lock(mutex,1);      // bloquea mutex
                        *(message_buffer_ptr)       = dumy;     // escribe valor en buffer
                        *(message_buffer_ptr_join)  = 0;        // obliga a vaciar la cache
                        *(message_buffer_ptr_dele)  = 0;        // obliga a vaciar la cache
                        altera_avalon_mutex_unlock(mutex);      // libera mutex
                        message_buffer_val          = dumy;
                }
        }
```

Benchmark para multiprocesador 2-núcleos: código fuente, Parte 4 (MV_paralelo_maestro.c)

CÓMPUTO

```
int main() {
…
// COMPUTO MAESTRO - Operacion Matriz x Vector
int local_n         = n / thread_count;
int my_first_row = rank * local_n;            // 1™ fila asignada a este nucleo
int my_last_row  = (rank+1) * local_n - 1;  // ultima fila asignada a este nucleo
for (k = 0; k < Niter; k++) {
                iteraciones++;
                for (i=my_first_row; i<=my_last_row; i++){
                                dumy = y[i];
                                for(j=0; j<m; j++){
                                                dumy += A[i*m+j] * x[j];
                                }
                                y[i] = dumy;
                }
}
```

SINCRONIZACIÓN DE UNIÓN (JOIN)

Benchmark para multiprocesador 2-núcleos: código fuente, Parte 5 (MV_paralelo_maestro.c)

```
// sincronizacion JOIN - barrera para unificación de hilos
message_buffer_val_join = 1;                          // indica que maestro ha llegado a JOIN
altera_avalon_mutex_lock(mutex,1);                    // bloquea mutex
*(message_buffer_ptr_join)   |= message_buffer_val_join; // inicializa RAM JOIN por parte del maestro
altera_avalon_mutex_unlock(mutex);                    // libera mutex


while( (message_buffer_val != 6) ){                                   // message_buffer_val = 6 : los dos hilos estan sincronizados en JOIN
        altera_avalon_mutex_lock(mutex,1);                            // bloquea mutex
        message_buffer_val      = *(message_buffer_ptr);              // lee valor en RAM
        message_buffer_val_join = *(message_buffer_ptr_join);         // lee valor en RAM
        altera_avalon_mutex_unlock(mutex);                            // libera mutex
        if ( (message_buffer_val_join == 0x3 && thread_count == 2 ) ||
                (message_buffer_val_join == 0x1 && thread_count == 1) ){
                dumy = 6;
                altera_avalon_mutex_lock(mutex,1);      // bloquea mutex
                *(message_buffer_ptr)   = dumy;         // escribe valor en RAM indicando que los hilos estan sincronizados JOIN
                altera_avalon_mutex_unlock(mutex);      // libera mutex
                message_buffer_val      = dumy;         // actualiza variable local
        }
}
```

Benchmark para multiprocesador 2-núcleos: código fuente. Parte 1 (MV_paralelo_esclavo.c)

CPU2 (núcleo 2)

```
int main(){
alt_mutex_dev* mutex = altera_avalon_mutex_open("/dev/message_buffer_mutex");
int rank                    = 1;        // hilo esclavo para nucleo= CPU2
int message_buffer_val      = 0x0;      // variable local, copia de variable global
int message_buffer_val_fork = 0x0;      // variable local, copia de variable global
int message_buffer_val_join = 0x0;      // variable local, copia de variable global
int thread_count            = 0;        // variable local, copia de variable global
int Niter                   = 0;        // variable local, copia de variable global
// FORK del Hilo-1, sincronizacion desde Hilo-0, message_buffer_val=15
while(message_buffer_val != 5) {
        altera_avalon_mutex_lock(mutex,2);          // bloquea mutex
        *(message_buffer_ptr_dele) = 0;             // obliga a vaciar la dCache de RAM sincronizacion
        message_buffer_val         = *(message_buffer_ptr);          // lee valor en buffer
        thread_count               = *(message_buffer_threads);
        Niter                      = *(message_buffer_Niter);
        if(message_buffer_val == 15 && thread_count == 2) {
                message_buffer_val_fork     = *(message_buffer_ptr_fork);   // lee valor en RAM
                message_buffer_val_fork     |= 2;
                *(message_buffer_ptr_fork)  = message_buffer_val_fork;
        }
        altera_avalon_mutex_unlock(mutex);          // libera mutex
}
```

SINCRONIZACION DE DISTRIBUCIÓN (FORK)

Benchmark para multiprocesador 2-núcleos: código fuente. Parte 2 (MV_paralelo_esclavo.c)

CPU2
(núcleo 2)

CÓMPUTO

```
int main(){
…
// COMPUTO ESCLAVO - Operacion matriz-vector
int local_n         = n / thread_count;
int my_first_row = rank * local_n;              // 1™ fila asignada a este nucleo
int my_last_row  = (rank+1) * local_n - 1;      // ultima fila asignada a este nucleo
for (k = 0; k < Niter; k++) {
                iteraciones++;
                for (i=my_first_row; i<=my_last_row; i++){
                                dumy = y[i];
                                for(j=0; j<m; j++){
                                                dumy += A[i*m+j] * x[j];
                                }
                                y[i] = dumy;
                }
}
```

Asignación de carga computacional

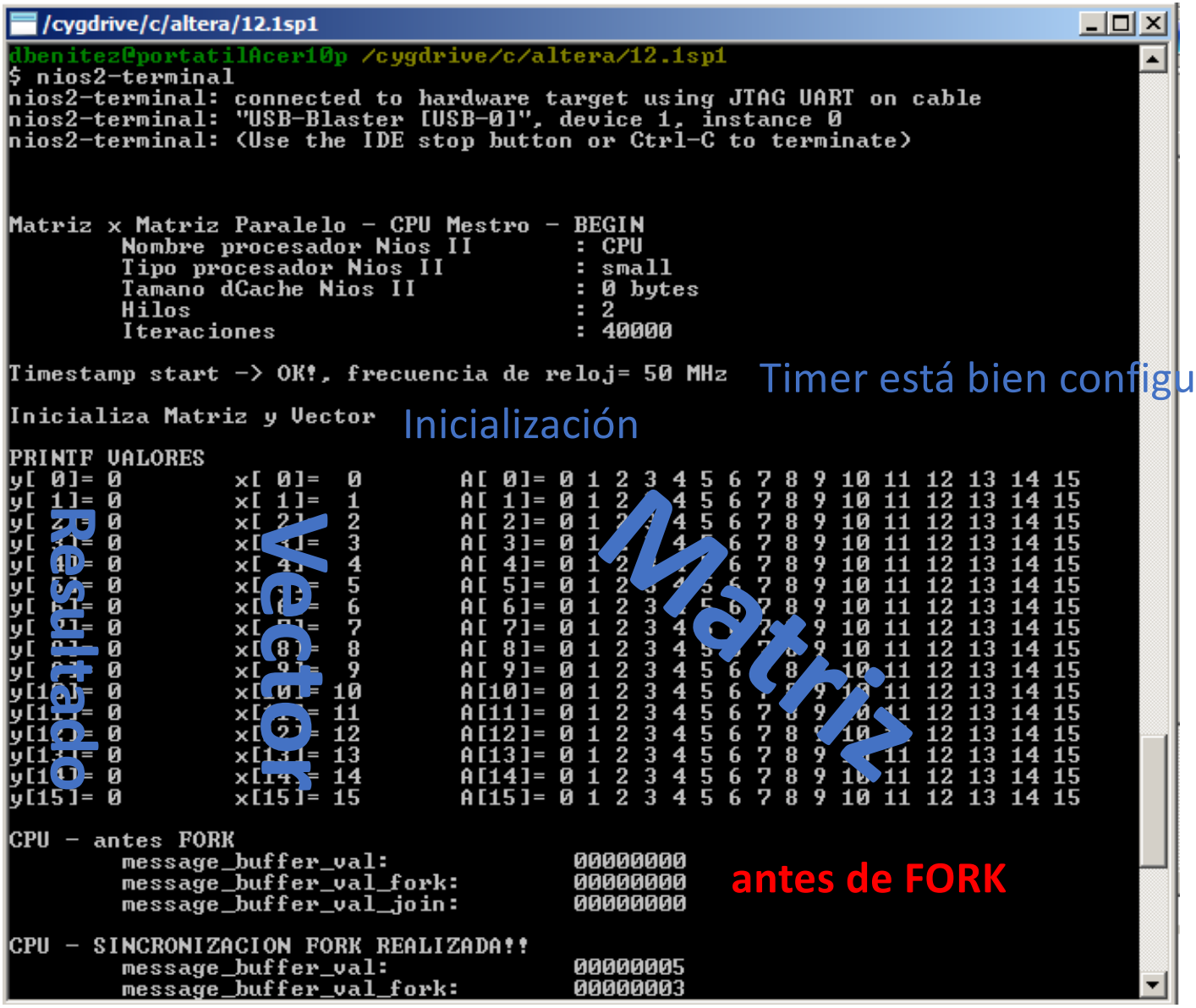
Benchmark para multiprocesador 2-núcleos: código fuente. Parte 3 (MV_paralelo_esclavo.c)

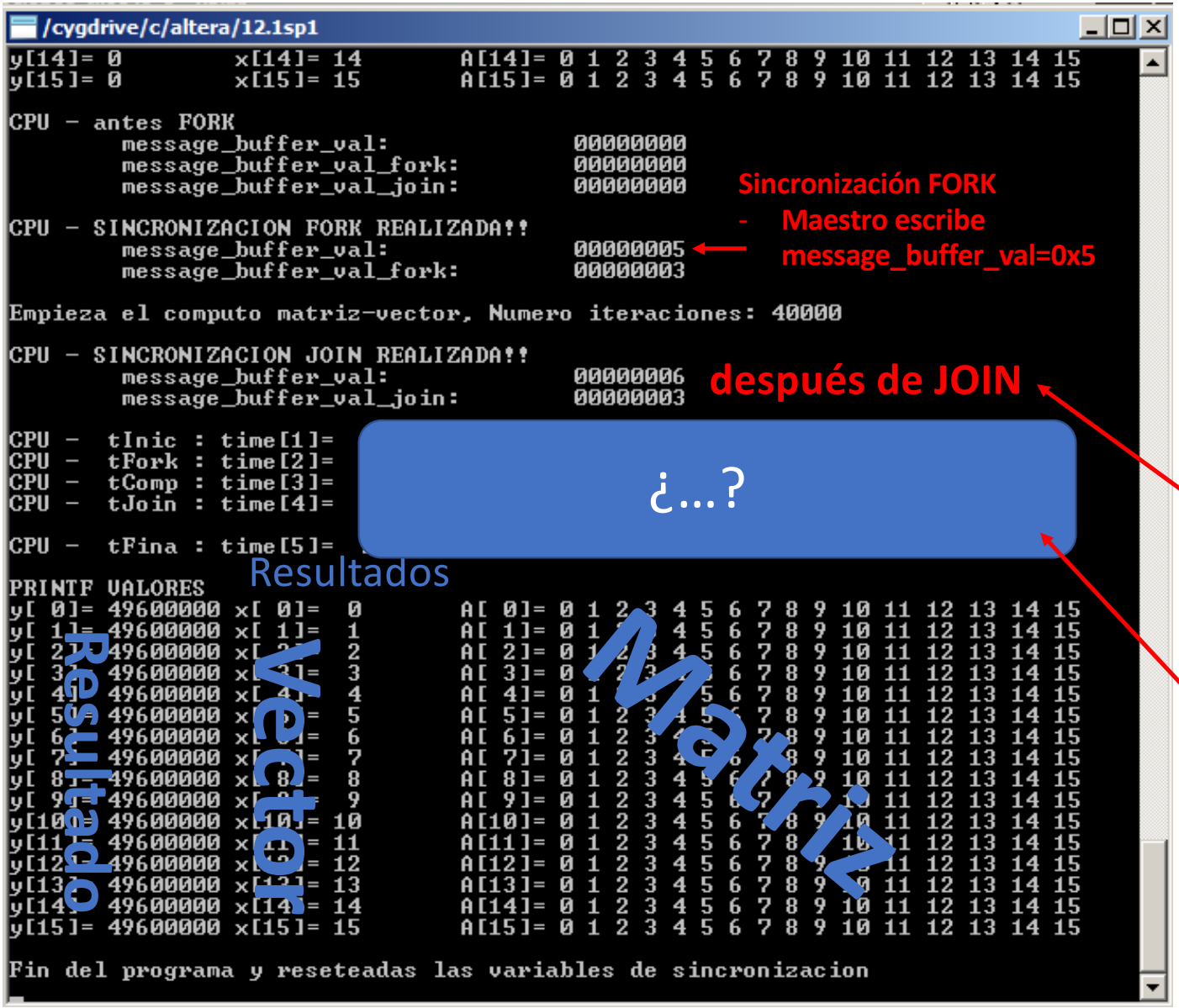
SINCRONIZACIÓN DE UNIÓN (JOIN)

CPU2
(núcleo 2)

```
int main(){
...
// JOIN - Unificacion de hilos
while(message_buffer_val != 6 && thread_count == 2) {
        altera_avalon_mutex_lock(mutex,2);                  // bloquea mutex
        message_buffer_val      = *(message_buffer_ptr);    // lee valor de variable compartida
        message_buffer_val_join = *(message_buffer_ptr_join); // lee valor de variable compartida
        message_buffer_val_join |= 2;                       // modifica la variable compartida para indicar que hilo esclavo ha llegado a JOIN
        *(message_buffer_ptr_join) = message_buffer_val_join;
        altera_avalon_mutex_unlock(mutex);                  // libera mutex
}

} // main()
```

Ejecución del programa paralelo en multiprocesador DualCoreNios2s





Ejecución del programa paralelo en multiprocesador DualCoreNios2s

Sincronización JOIN
- Maestro escribe message_buffer_val=0x1
- Esclavo escribe : message_buffer_val | 0x2
- Termina el JOIN cuando message_buffer_val=0x6

Tiempos de ejecución

Evaluación de prestaciones del multiprocesador Nios II, versión Secuencial (MV_serie.c)

- **Objetivo 3-1**:
  - Registrar tiempos en CPU para 4 cargas de trabajo. Parámetros:
    - Niter = 1000, 2000, 5000, 10000.
  - Evaluación de prestaciones (tiempos de ejecución): realizar Tabla 1 (apuntar solo el tiempo total)
    - Cómputo: tiempo invertido en la multiplicación Matriz x Vector
    - Total: tiempo de ejecución del programa completo
  - Repetir los resultados para:
    - Nios II/e (configuración FPGA: DualCoreNios2e)
    - Nios II/s (configuración FPGA: DualCoreNios2s)
  - Justificar resultados:
    - ¿Es razonable que el doble de iteraciones (Niter) ocasione que el tiempo de ejecución de uno de los procesadores Nios II/e del multiprocesador DualCoreNios2e sea el doble?
    - ¿Por qué?

Evaluación de prestaciones de los núcleos procesadores Nios II/{e,s}

Tabla 1. Registros de los tiempos de ejecución del algoritmo secuencial Matriz x Vector para uno de los núcleos procesador (CPU) de dos multiprocesadores Nios II.

Configuración FPGA	Niter	Tiempo total (ms)	Speed-up
DualCoreNios2e	1000		1
DualCoreNios2e	2000		1
DualCoreNios2e	5000		1
DualCoreNios2e	10000		1
DualCoreNios2s	1000		
DualCoreNios2s	2000		
DualCoreNios2s	5000		
DualCoreNios2s	10000		

Multiprocesador Nios II/**e**

Multiprocesador Nios II/**s**

Ejecución **secuencial en un núcleo procesador**

Evaluación de prestaciones de los multiprocesadores de 2-núcleos Nios II/{e,s}

- **Objetivo 3-2** – Ejecutar y evaluar las prestaciones del algoritmo Matriz x Vector utilizando dos núcleos de procesador Nios II:
  - Ejecutar con multiprocesador **DualCoreNios2e** (doble núcleo: 2 x Nios II/e)
  - Registrar tiempos activando un solo hilo, usando el núcleo CPU del multiprocesador para 4 cargas de trabajo. Parámetros:
    - Nthreads = 1
    - Niter = 1000, 2000, 5000, 10000.
  - Registrar tiempos activando dos hilos, usando los núcleos CPU y CPU2 del multiprocesador para 4 cargas de trabajo. Parámetros:
    - Nthreads = 2
    - Niter = 1000, 2000, 5000, 10000.
  - Evaluación de prestaciones: realizar Tabla 2 (descripción de la tabla en transparencia siguiente)
  - Repetir los resultados con multiprocesador **DualCoreNios2s** (doble núcleo: 2 x Nios II/s)
  - Justificar resultados: ¿los resultados son similares al Objetivo 3-1?, ¿por qué?

Evaluación de prestaciones de los multiprocesadores de 2-núcleos Nios II/{e,s}

- **Objetivo 3-2** – Ejecutar y evaluar las prestaciones del algoritmo Matriz x Vector utilizando dos núcleos de procesador Nios II:
  - Evaluación de prestaciones: realizar Tabla 2
    - Tiempo Fork: tiempo de la sincronización FORK
    - Tiempo Cómputo: tiempo invertido en la multiplicación Matriz x Vector
    - Tiempo Join: tiempo de la sincronización JOIN
    - Tiempo Total: tiempo de ejecución del programa completo
    - $\text{Speed-Up}_{\text{Computo}}$: $\text{Tiempo}_{\text{Computo-1hilo}}$ / $\text{Tiempo}_{\text{Computo-2hilo}}$
    - $\text{Speed-Up}_{\text{Total}}$: $\text{Tiempo}_{\text{Total-1hilo}}$ / $\text{Tiempo}_{\text{Total-2hilo}}$
    - $\text{EficienciaParalelismo}_{\text{Computo}}$: 100,0% × $\text{Speed-Up}_{\text{Computo}}$ / 2,0
    - $\text{EficienciaParalelismo}_{\text{Total}}$: 100,0% × $\text{Speed-Up}_{\text{Total}}$ / 2,0

Evaluación de prestaciones de los núcleos procesadores Nios II/{e,s}

Tabla 2. Evaluación de prestaciones del algoritmo Matriz x Vector para las versiones de 1 y 2 hilos usando los núcleos procesadores CPU y CPU2 de dos multiprocesadores Nios II

Configuración FPGA	Hilos	Niter	Tiempo (ms)				Speed-up		Eficiencia paralelismo	
			Fork	Cómputo	Join	Total	Cómputo	Total	Cómputo	Total
DualCoreNios2e	1	1000					1	1	100%	100%
DualCoreNios2e	1	2000					1	1	100%	100%
DualCoreNios2e	1	5000					1	1	100%	100%
DualCoreNios2e	1	10000					1	1	100%	100%
DualCoreNios2e	2	1000								
DualCoreNios2e	2	2000								
DualCoreNios2e	2	5000								
DualCoreNios2e	2	10000								
DualCoreNios2s	1	1000					1	1	100%	100%
DualCoreNios2s	1	2000					1	1	100%	100%
DualCoreNios2s	1	5000					1	1	100%	100%
DualCoreNios2s	1	10000					1	1	100%	100%
DualCoreNios2s	2	1000								
DualCoreNios2s	2	2000								
DualCoreNios2s	2	5000								
DualCoreNios2s	2	10000								

Multiprocesador Nios II/**e**

Multiprocesador Nios II/**s**

Ejecución **paralela 1-hilo**

Ejecución **paralela 2-hilo**

Ejecución **paralela 1-hilo**

Ejecución **paralela 2-hilo**

Evaluación de prestaciones del multiprocesador Nios II/{e,s} con benchmark Matriz x Matriz

- **Objetivo 3-3** – Crear, ejecutar y evaluar las prestaciones de un benchmark Matriz x Matriz (C[] = A[] x B[]) que realiza la multiplicación de 2 matrices:
  - Codificar la multiplicación de matrices de tamaño 8 x 8
  - Ejecutar con DualCoreNios2e (2 núcleos: 2 x Nios II/e) y DualCoreNios2s (2 núcleos: 2 x Nios II/s). Parámetros:
    - thread_count =1, 2.
    - Niter = 1000, 2000, 5000, 10000.
  - Evaluación de prestaciones: realizar una tabla similar a la Tabla 2 para el algoritmo Matriz x Vector
  - Justificar resultados: ¿los resultados son similares al Objetivo 3-2?, ¿por qué?

```
void Matriz-Matriz (int n, int* A, int* B, int* C){
 int i,j,k,cij;
 for (i = 0; i < n; ++i)          /* i: fila de la matriz */
   for (j = 0; j < n; ++j)  {     /* j: columna de la matriz */
    cij = C[i*n+j];               /* cij ← C[i][j] */
    for ( k = 0; k < n; k++ )     /* cij: fila-A(i) x columna-B(j) */
     cij += A[i*n+k] * B[k*n+j]; /* cij += A[i][k]*B[k][j] */
    C[i*n+j] = cij;               /* C[i][j] ← cij */
   }
}
```

Trabajo opcional – Objetivo 3-4

- Realizar la implementación del algoritmo Matriz x Vector usando el multiprocesador basado en Nios II/f. Usar matrices 16 x 16.
- Realizar la implementación del algoritmo Matriz x Matriz usando el multiprocesador basado en Nios II/f. Usar matrices 8 x 8.
- Evaluar las prestaciones del multiprocesador Nios II/f usando los tiempos de ejecución, speed-up y eficiencia del paralelismo cuando el número de repeticiones de las correspondientes operaciones matriciales son: 1000, 2000, 5000, 10000.
- Comparar las prestaciones de los tres procesadores Nios II/{e,s,f} en base a las implementaciones de los algoritmos matrix x vector y matriz x matriz.
- Observación. Nios II/f tiene cache de datos y por ello, es necesario forzar la actualización de la memoria principal después de que un determinado hilo actualice una variable compartida.

Matriz x Vector, Nios II/f

```
/cygdrive/c/altera/12.1sp1

dbenitez@portatilAcer10p /cygdrive/c/altera/12.1sp1
$ nios2-terminal
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)



Matriz x Vector Paralelo - CPU Mestro Nios II/f - BEGIN
        Nombre procesador Nios II       : CPU
        Tipo procesador Nios II         : fast
        Tamano dCache Nios II           : 4096 bytes
        Hilos                           : 2
        Iteraciones                     : 30000

Timestamp start -> OK!, frecuencia de reloj= 50 MHz

Inicializa Matriz y Vector

FLUSHmaestro de toda dCache

PRINTF VALORES

FLUSHmaestro de toda dCache
y[ 0]= 0        x[ 0]=  0       A[ 0]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 1]= 0        x[ 1]=  1       A[ 1]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 2]= 0        x[ 2]=  2       A[ 2]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 3]= 0        x[ 3]=  3       A[ 3]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 4]= 0        x[ 4]=  4       A[ 4]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 5]= 0        x[ 5]=  5       A[ 5]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 6]= 0        x[ 6]=  6       A[ 6]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 7]= 0        x[ 7]=  7       A[ 7]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 8]= 0        x[ 8]=  8       A[ 8]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 9]= 0        x[ 9]=  9       A[ 9]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[10]= 0        x[10]= 10       A[10]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[11]= 0        x[11]= 11       A[11]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[12]= 0        x[12]= 12       A[12]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[13]= 0        x[13]= 13       A[13]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[14]= 0        x[14]= 14       A[14]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[15]= 0        x[15]= 15       A[15]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

```
/cygdrive/c/altera/12.1sp1

CPU - antes FORK
        message_buffer_val:             00000000
        message_buffer_val_fork:        00000000
        message_buffer_val_join:        00000000

CPU - SINCRONIZACION FORK REALIZADA!!
        message_buffer_val:             00000005
        message_buffer_val_fork:        00000003

Empieza el computo matriz-vector, Numero iteraciones: 30000

CPU - SINCRONIZACION JOIN REALIZADA!!
        message_buffer_val:             00000006
        message_buffer_val_join:        00000003

CPU -  tInic : time[1]=    4646707 clk   (    89 ms) intervalo=     89 ms
CPU -  tFork : time[2]=    4847282 clk   (    93 ms) intervalo=      4 ms
CPU -  tComp : time[3]=   66372867 clk   (  1324 ms) intervalo=   1230 ms
CPU -  tJoin : time[4]=   77078809 clk   (  1538 ms) intervalo=    214 ms

CPU -  tFina : time[5]=   77078809 clk   TiempoTotal=   1538 ms

PRINTF VALORES

FLUSHmaestro de toda dCache
y[ 0]= 37200000 x[ 0]=  0       A[ 0]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 1]= 37200000 x[ 1]=  1       A[ 1]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 2]= 37200000 x[ 2]=  2       A[ 2]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 3]= 37200000 x[ 3]=  3       A[ 3]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 4]= 37200000 x[ 4]=  4       A[ 4]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 5]= 37200000 x[ 5]=  5       A[ 5]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 6]= 37200000 x[ 6]=  6       A[ 6]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 7]= 37200000 x[ 7]=  7       A[ 7]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 8]= 37200000 x[ 8]=  8       A[ 8]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[ 9]= 37200000 x[ 9]=  9       A[ 9]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[10]= 37200000 x[10]= 10       A[10]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[11]= 37200000 x[11]= 11       A[11]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[12]= 37200000 x[12]= 12       A[12]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[13]= 37200000 x[13]= 13       A[13]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[14]= 37200000 x[14]= 14       A[14]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
y[15]= 37200000 x[15]= 15       A[15]= 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Fin del programa y reseteadas las variables de sincronizacion
```

Implementación Matriz x Matriz

DualCoreNios2e

- Ficheros *.c
  - MM_maestro3_nios2e.c
  - MM_esclavo3_nios2e.c

DualCoreNios2s

- Ficheros *.c
  - MM_maestro3_nios2s.c
  - MM_esclavo3_nios2s.c

DualCoreNios2f_dCache4K

- Ficheros *.c
  - MM_maestro3_nios2f.c
  - MM_esclavo3_nios2f.c
- Es necesario borrar la cache de datos de cada procesador con las variables que escribe el otro procesador. Variables:
  - Matrices de entrada: A,B
  - Matriz resultante: C
  - Variables fork-join: message_buffer_ptr , message_buffer_ptr_fork, message_buffer_ptr_join

Matriz x Matriz, DualCoreNios2e

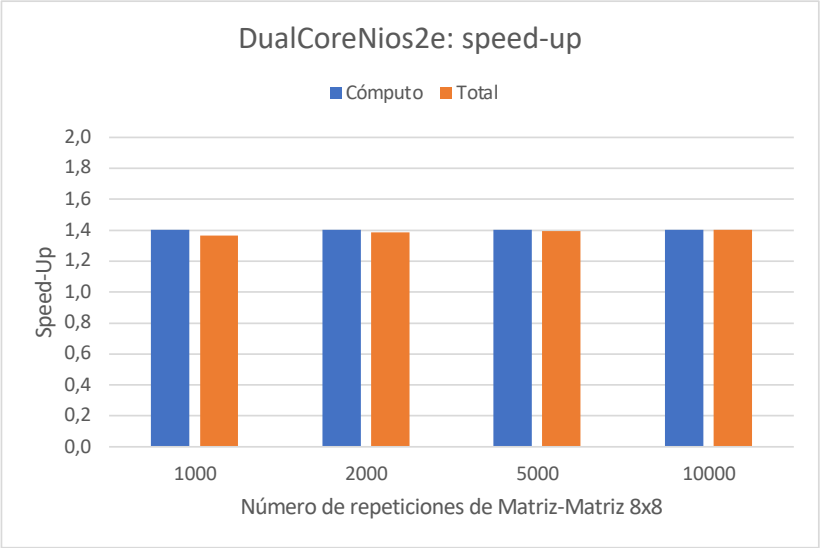
- Evaluación de prestaciones

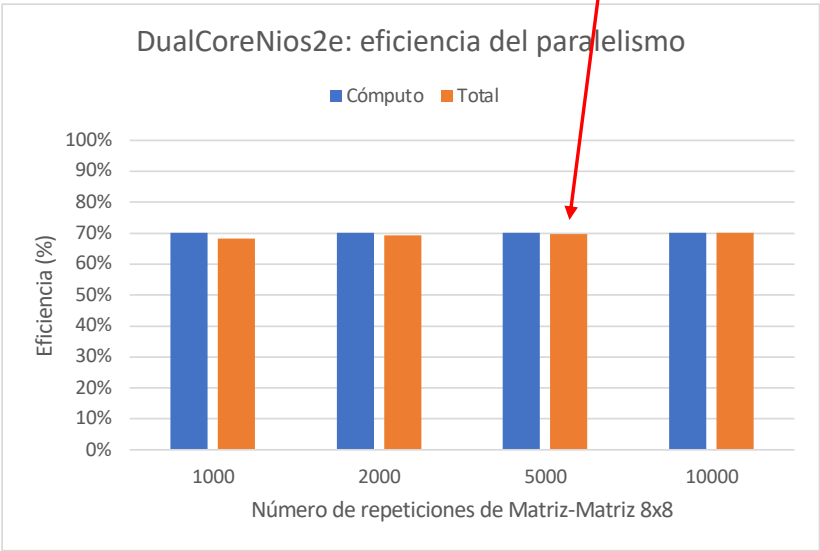
			Tiempo (ms)	Tiempo (ms)	Tiempo (ms)	Tiempo (ms)	Speed-up	Speed-up	Eficiencia del paralelismo	
Multiprocesador			Fork	Cómputo	Join	Total	Cómputo	Total	Cómputo	Total
DualCoreNios2e	1	1000	2	4574	254	4848	1,00	1,00		
DualCoreNios2e	1	2000	2	9145	169	9337	1,00	1,00		
DualCoreNios2e	1	5000	2	22857	236	23117	1,00	1,00		
DualCoreNios2e	1	10000	5	45710	179	45912	1,00	1,00		
DualCoreNios2e	1	20000	4	85872	0	85895				
DualCoreNios2e	1	30000	2	85873	0	85896				
DualCoreNios2e	1	40000	5	85867	0	85895				
DualCoreNios2e	**2**	1000	5	3259	263	3550	1,40	1,37	70,2%	68,3%
DualCoreNios2e	**2**	2000	5	6519	189	6737	1,40	1,39	70,1%	69,3%
DualCoreNios2e	**2**	5000	6	16283	249	16561	1,40	1,40	70,2%	69,8%
DualCoreNios2e	**2**	10000	3	32562	128	32719	1,40	1,40	70,2%	70,2%
DualCoreNios2e	**2**	20000	3	65123	128	65270				
DualCoreNios2e	**2**	30000	5	85866	0	85895				
DualCoreNios2e	**2**	40000	5	85867	0	85894				

Matriz x Matriz, DualCoreNios2e

- Evaluación de prestaciones

Se consigue un máximo de **70%** de eficiencia con procesadores **Nios II/e**





Matriz x Matriz, DualCoreNios2s

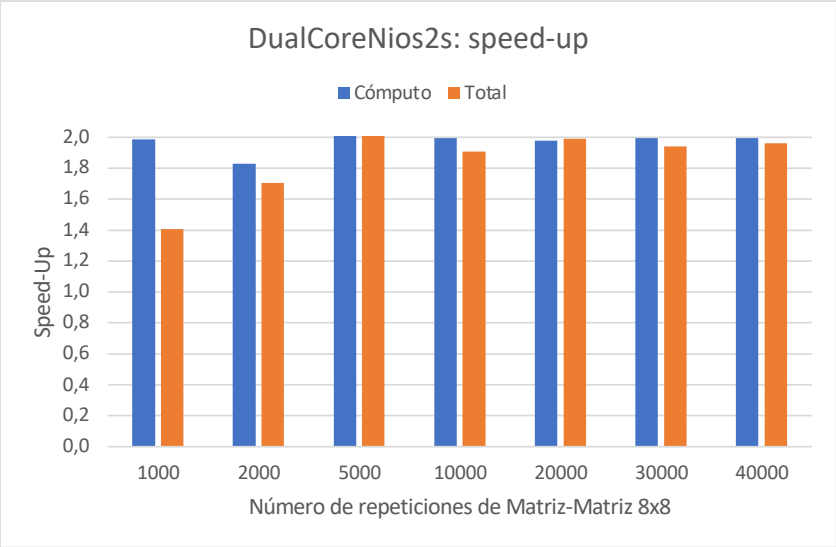
- Evaluación de prestaciones

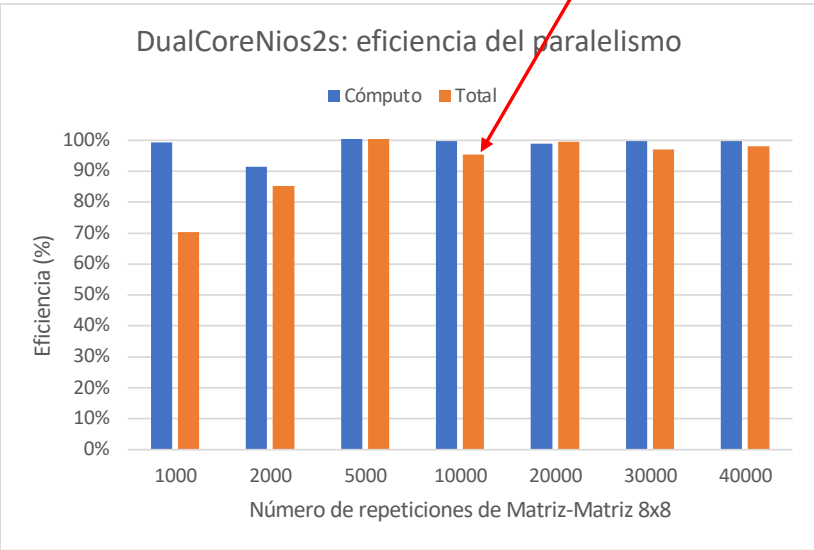
			Tiempo (ms)	Tiempo (ms)	Tiempo (ms)	Tiempo (ms)	Speed-up	Speed-up	Eficiencia del paralelismo	
Multiprocesador			Fork	Cómputo	Join	Total	Cómputo	Total	Cómputo	Total
DualCoreNios2s	1	1000	0	516	109	634	1,00	1,00		
DualCoreNios2s	1	2000	0	1032	93	1133	1,00	1,00		
DualCoreNios2s	1	5000	0	2581	347	2936	1,00	1,00		
DualCoreNios2s	1	10000	0	5161	69	5239	1,00	1,00		
DualCoreNios2s	1	20000	0	10232	103	10434	1,00	1,00		
DualCoreNios2s	1	30000	0	15484	154	15646	1,00	1,00		
DualCoreNios2s	1	40000	0	20645	80	20733	1,00	1,00		
DualCoreNios2s	**2**	1000	1	260	170	451	1,98	1,41	99,2%	70,3%
DualCoreNios2s	**2**	2000	1	564	76	664	1,83	1,71	91,5%	85,3%
DualCoreNios2s	**2**	5000	1	1281	145	1445	2,01	2,03	100,7%	101,6%
DualCoreNios2s	**2**	10000	1	2587	139	2745	1,99	1,91	99,7%	95,4%
DualCoreNios2s	**2**	20000	1	5174	55	5247	1,98	1,99	98,9%	99,4%
DualCoreNios2s	**2**	30000	1	7761	275	8057	2,00	1,94	99,8%	97,1%
DualCoreNios2s	**2**	40000	1	10344	199	10564	2,00	1,96	99,8%	98,1%

Matriz x Matriz, DualCoreNios2s

- Evaluación de prestaciones

No se consigue un máximo de **100%** de eficiencia en el tiempo total con procesadores **Nios II/s** para todos los volúmenes de datos debido a los tiempos de sincronización FORK y JOIN





Matriz x Matriz, DualCoreNios2f_dCache4K

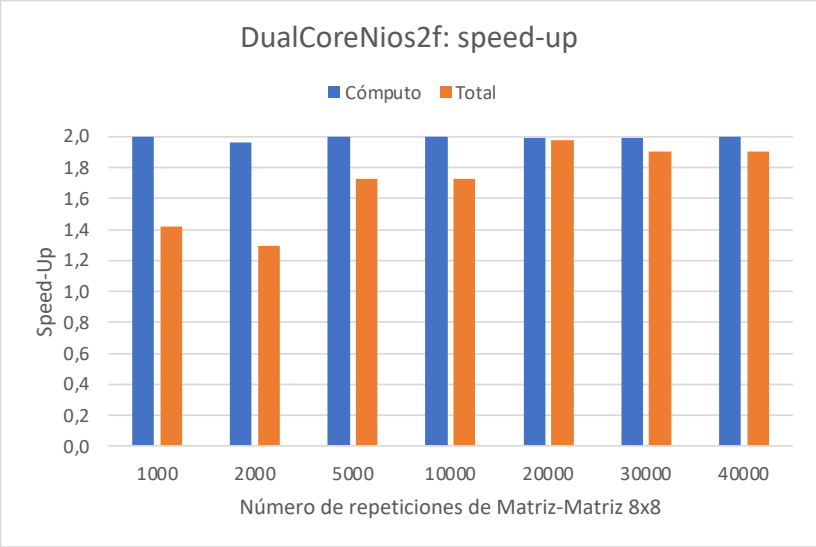
- Evaluación de prestaciones

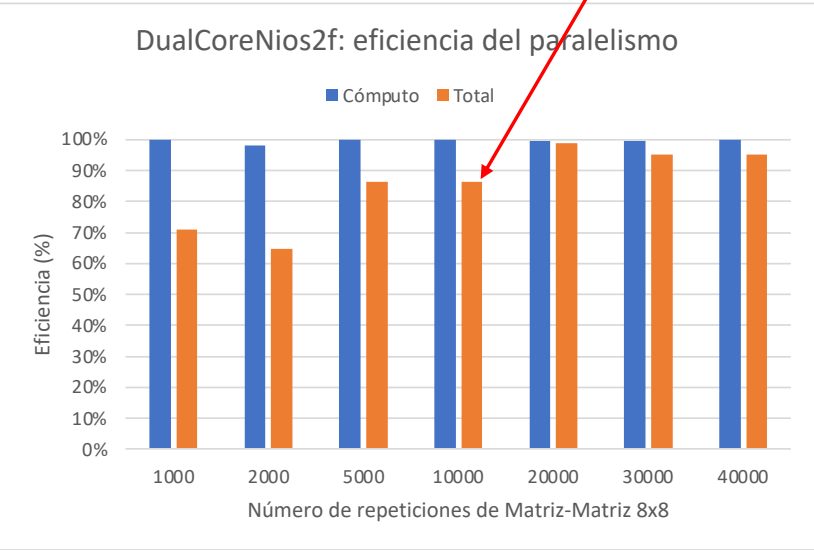
			Tiempo (ms)	Tiempo (ms)	Tiempo (ms)	Tiempo (ms)	Speed-up	Speed-up	Eficiencia del paralelismo	
Multiprocesador			Fork	Cómputo	Join	Total	Cómputo	Total	Cómputo	Total
DualCoreNios2f	1	1000	0	162	196	368	1,00	1,00		
DualCoreNios2f	1	2000	0	324	141	475	1,00	1,00		
DualCoreNios2f	1	5000	0	810	201	1021	1,00	1,00		
DualCoreNios2f	1	10000	0	1622	150	1782	1,00	1,00		
DualCoreNios2f	1	20000	0	3239	288	3537	1,00	1,00		
DualCoreNios2f	1	30000	0	4859	190	5059	1,00	1,00		
DualCoreNios2f	1	40000	0	6481	211	6702	1,00	1,00		
DualCoreNios2f	**2**	1000	0	81	168	259	2,00	1,42	100,0%	71,0%
DualCoreNios2f	**2**	2000	0	165	191	366	1,96	1,30	98,2%	64,9%
DualCoreNios2f	**2**	5000	0	405	175	590	2,00	1,73	100,0%	86,5%
DualCoreNios2f	**2**	10000	0	811	211	1031	2,00	1,73	100,0%	86,4%
DualCoreNios2f	**2**	20000	0	1624	155	1789	1,99	1,98	99,7%	98,9%
DualCoreNios2f	**2**	30000	0	2435	209	2654	2,00	1,91	99,8%	95,3%
DualCoreNios2f	**2**	40000	0	3243	269	3525	2,00	1,90	99,9%	95,1%

Matriz x Matriz, DualCoreNios2f_dCache4K

- Evaluación de prestaciones

No se consigue un máximo de **100%** de eficiencia en el tiempo total con procesadores **Nios II/s** para todos los volúmenes de datos debido a los tiempos de sincronización FORK y JOIN





Matriz x Matriz

- Evaluación de prestaciones: comparación entre procesadores Nios II

Speed-Up Cómputo 1 hilo				Speed-Up Total 1 hilo		
	Baseline				Baseline	
	Nios2e	Nios2s			Nios2e	Nios2s
Nios2s	8,86			Nios2s	8,13	
Nios2f	27,87	3,18		Nios2f	21,10	2,72
Speed-Up Cómputo 2 hilos				Speed-Up Total 2 hilos		
	Baseline				Baseline	
	Nios2e	Nios2s			Nios2e	Nios2s
Nios2s	12,40			Nios2s	10,77	
Nios2f	40,04	3,22		Nios2f	25,68	2,52

Máximo speed-up de los procesadores Nios II/s y Nios II/f respecto a Nios II/e