JCSD 2015

Arquitectura Epiphany III. Una primera toma de contacto

S. Gálvez, F.J. Esteban, P. Hernández, G. Dorado

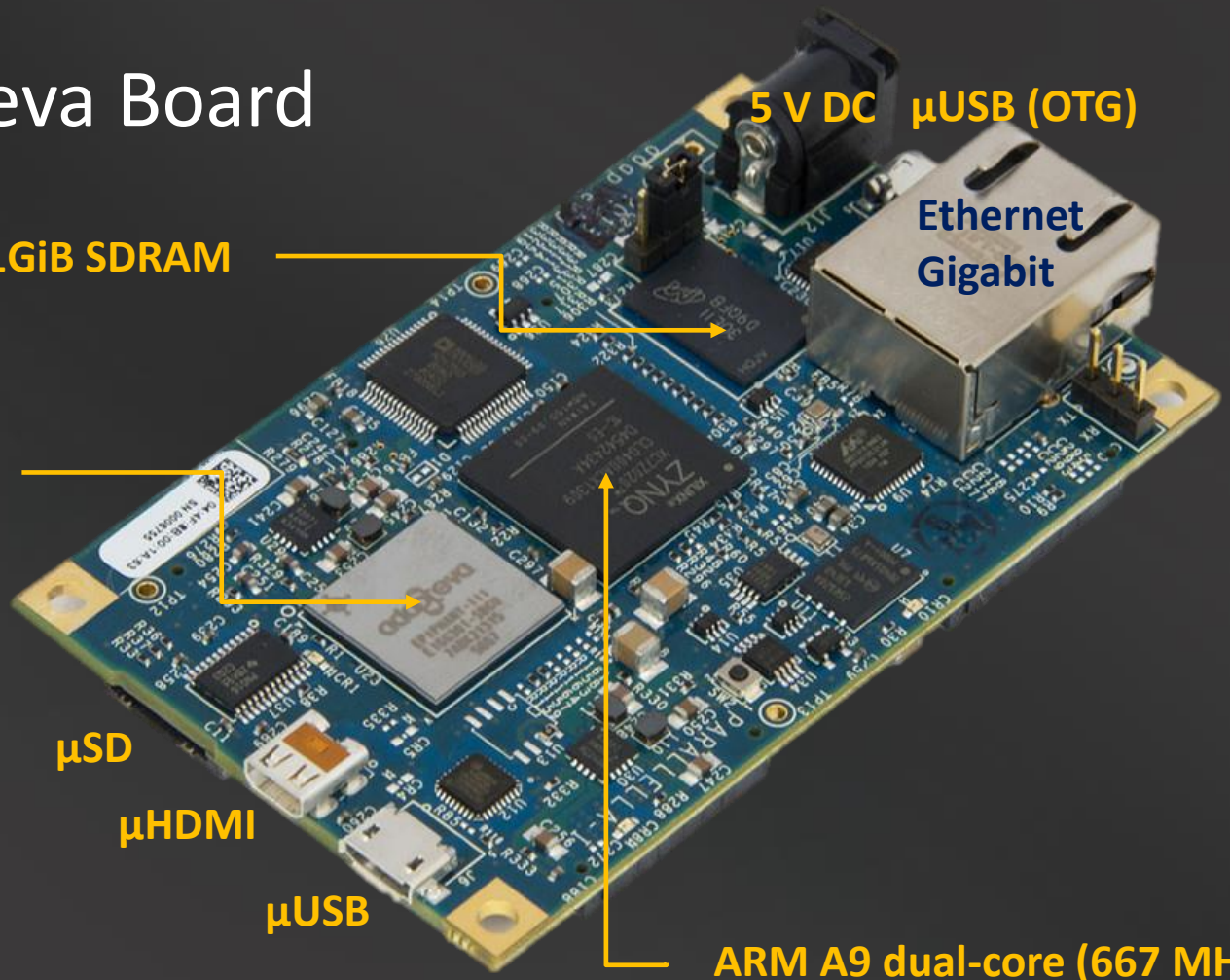
UNIVERSIDAD DE MÁLAGA | UNIVERSIDAD DE CÓRDOBA | CSIC | IAS

Contenido

- Arquitectura hardware
- Compilación, Ejecución y Mapa de memoria
- Entorno de desarrollo
- Caso práctico
  - Diseño
  - Implementación
  - Resultados
- Conclusiones

Arquitectura hardware

- Adapteva Board

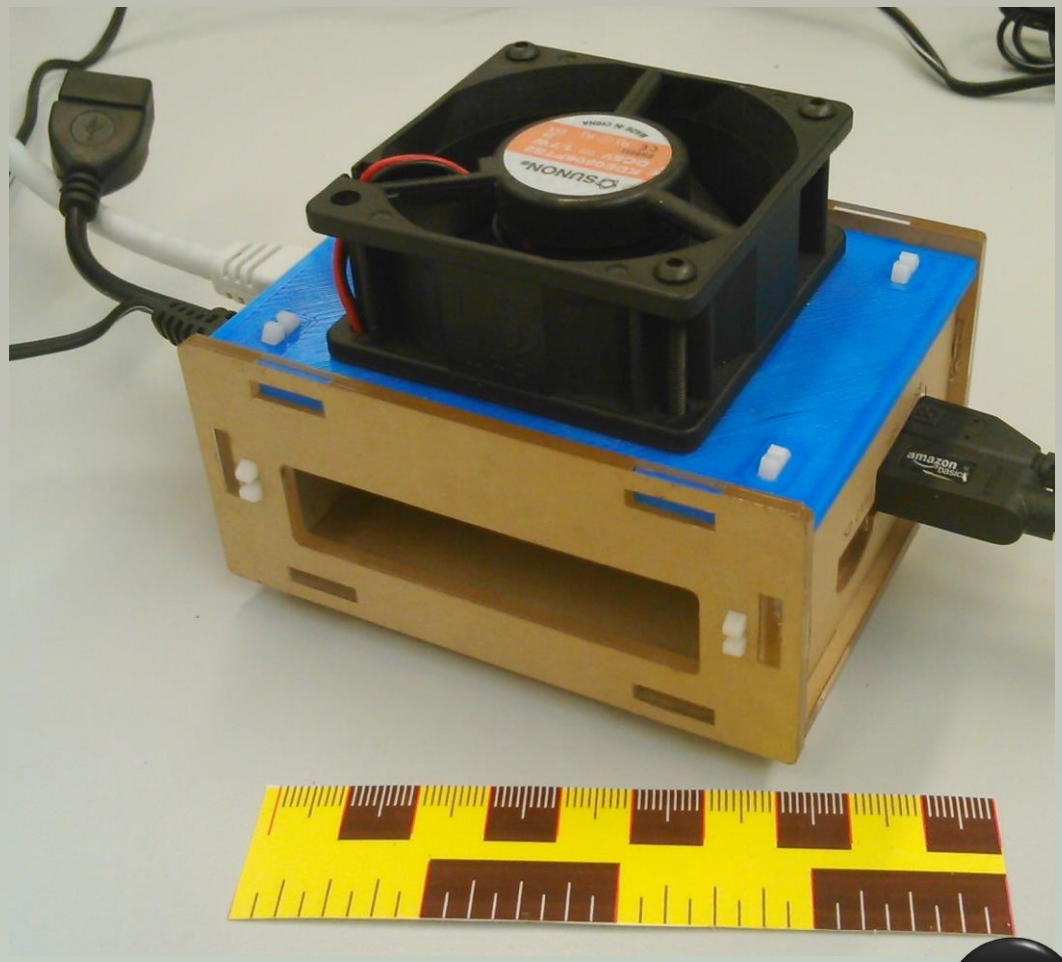


Características hardware

	E16G301 – Parallella board
Número de núcleos	16
Frecuencia de CPU	600MHz (modelo SBCU)
Tecnología de integración	65 nm
Arquitectura de los núcleos	RISC (32 bits)
Memoria cache	32 KiB de memoria local por núcleo
Soporte a la vectorización	Sin soporte
Soporte de coma flotante	Simple precisión
RAM en la placa host	1 GiB DDR3L
Consumo en vatios	0,7 vatios (~2 vatios la tarjeta)
Conexión a PC	Ethernet 1 Gbs
Entorno de desarrollo	No disponible

Puesta en funcionamiento

- Disipador térmico
- Caja
- Ventilador
- Arranque en SD
- Alimentación
- Teclado, ratón y monitor

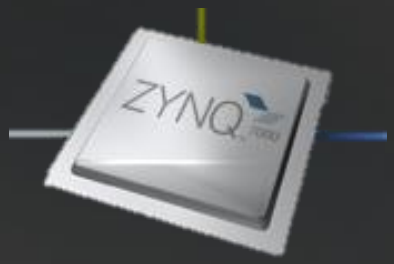


Compilación y Ejecución

- Compilación cruzada
- Arranque en host Zynq
- Sin E/S en E16G301

Desarrollo
sin soporte

1



Compilación
ARM y Epiphany

2

Ejecución y transferencia

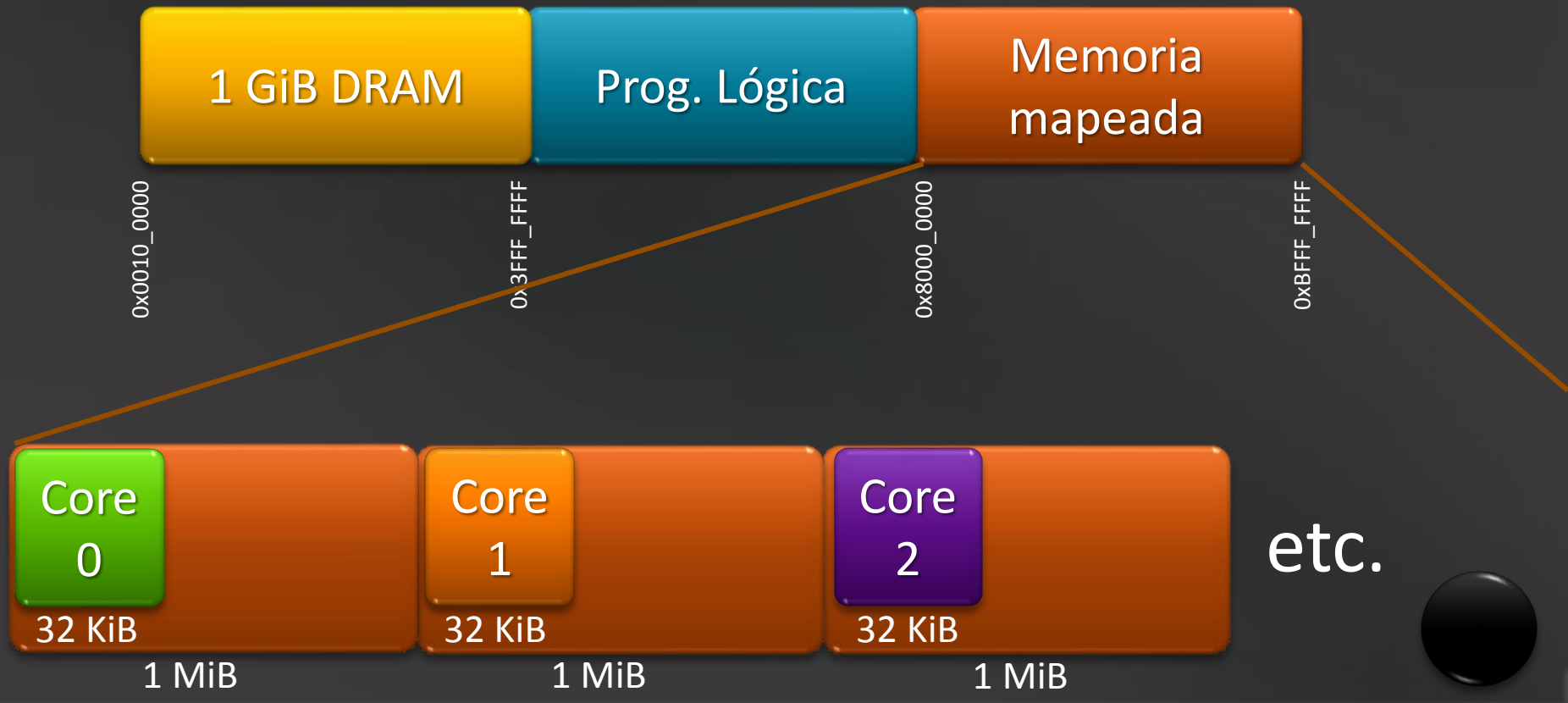
Ejecución

3

NFS

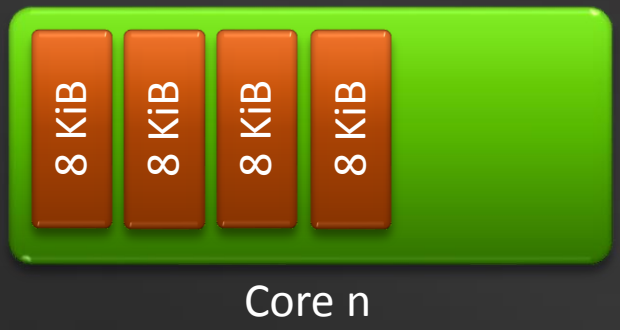
Mapa de memoria host

- Los 32 KiB de cada core están mapeados en memoria del host
- Así carga el host los programas en el E16G301



Mapa de memoria Epiphany

- Los 32 KiB se dividen en bloques de 8KiB



- Hay 32 MiB de memoria compartida como medio de comunicación host/Epiphany

Entorno de desarrollo

- Sin sincronización host-Epiphany
- API básica inter-core
  - Mutex
  - Barreras
- Necesidad de comunicación vía DMA
  - Problemas con volatile
- Uso de memoria dinámica desaconsejado
  - Gestión a realizar por el desarrollador con arrays
- No hay IDE

Caso práctico
Algoritmo Smith-Waterman con *affine gaps*

- Usado en Biología para buscar cadenas de ADN o de proteínas similares a una dada (*query*)
  - Busca en una base de cadenas
  - Compara la cadena *query* con todas y cada una
  - Obtiene una puntuación en base a una matriz
  - Afina el resultado introduciendo huecos si es necesario (mutaciones)
- Se usa programación dinámica

Caso práctico

Algoritmo Smith-Waterman (*affine gaps*). Ejemplo

```
Query:ATTCGTCC
Target: TTGGAATCC
```

```
Mismatch: -4
Open gap: 10
Extend gap: 1
```

Matriz:

	A	T	G	C
A	5	-4	-4	-4
T	-4	5	-4	-4
G	-4	-4	5	-4
C	-4	-4	-4	5

Alineamiento resultante (puntuación 3 –muy baja–):

```
ATTCG--TCC
-TTGGAATCC
```

Caso práctico

Algoritmo Smith-Waterman con *affine gaps*

- Dadas dos secuencias A y B
- Inicialización
  - H[i,0] = H[0,j] = E[0,j] = F[i,0] = 0
- Recurrencia:
  - E[i,j] = max(E[i-1,j]-ext, H[i-1,j]-open)
  - F[i,j] = max(F[i,j-1]-ext, H[i,j-1]-open)
  - H[i,j] = max(H[i-1,j-1]+M[A[i],B[j]], E[i,j], F[i,j])
- Se busca superar un umbral T

Caso práctico

Implementación

- I
- P
- T

e_sw_epi.c (/epiOfs/sw_epi/src) - gedit

Fichero Editar Ver Buscar Herramientas Documentos Ayuda

Abrir Guardar Deshacer

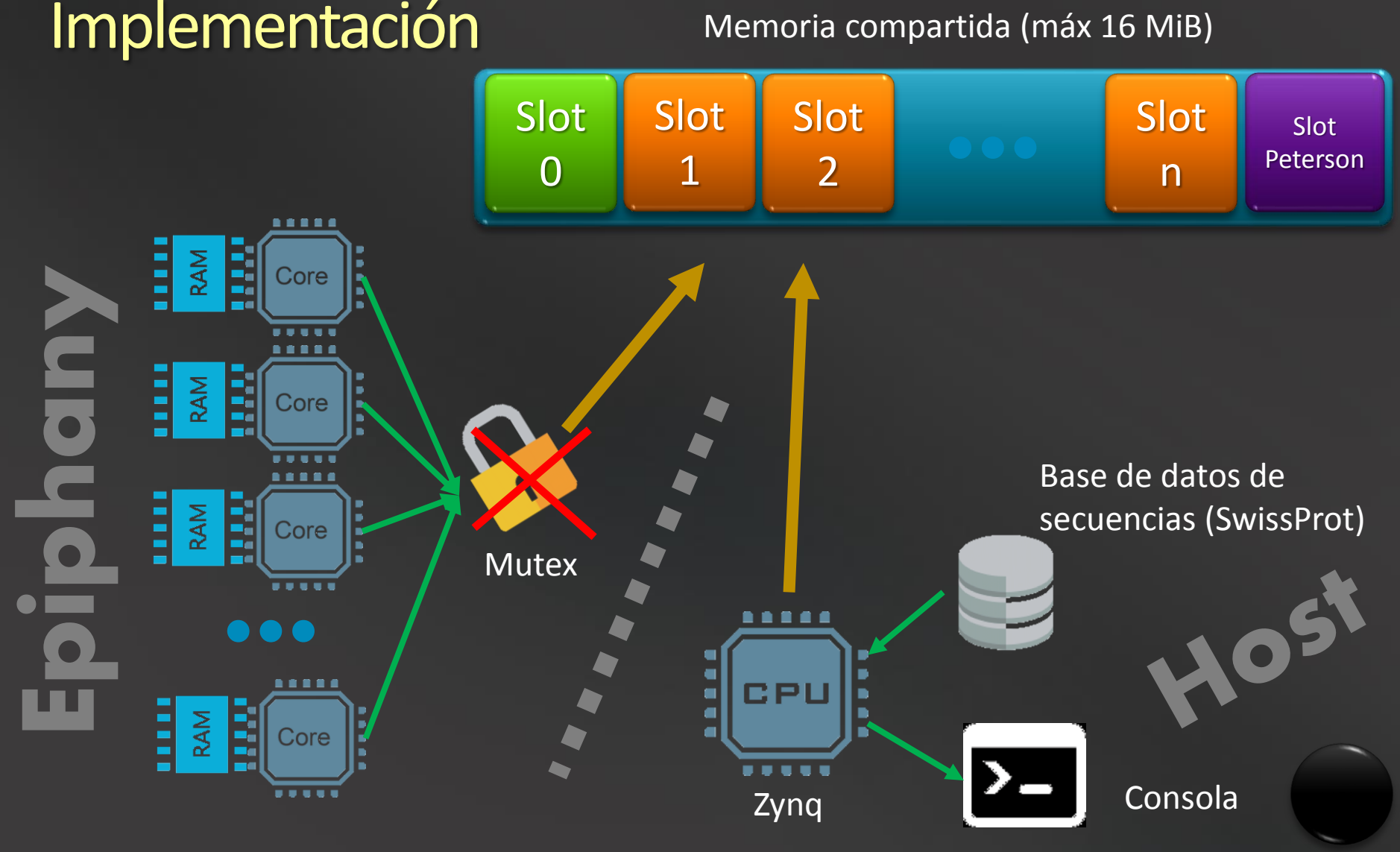
e_sw_epi.c

```
/*
 *   Estructura:
 *              estado de cores = 4*NUM_CORES bytes
 *              long_sec_ref = 4 bytes
 *              sec_ref = long_sec_ref bytes + necesario para alinear a palabra
 *              num_letras = 4 bytes
 *              letras = 128 bytes
 *              matriz = num_letras * num_letras bytes + necesario para alinear a palabra
 *              gap_open = 4 byte
 *              gap_extend = 4 byte
 *              umbral = 4 bytes
 */
void precarga_datos_algoritmo(){
        int indice_actual __attribute__((aligned(4)));
        uint32_t word_aux __attribute__((aligned(4)));
        indice_actual = 4*NUM_CORES;
        // Carga de la longitud de la secuencia de referencia
        e_dma_copy(&word_aux, (void *)buff+indice_actual, sizeof(word_aux));
        e_dma_wait(canal_dma);
        long_sec_ref = (int16_t)word_aux;
        // Carga de la secuencia de referencia. Se leen bytes de más para leer palabras completas.
        indice_actual += 4;
        e_dma_copy(local_buff, (void *)buff+indice_actual, sig_multiplo_4(long_sec_ref));
        e_dma_wait(canal_dma);
        sec_ref = local_buff;
        // Espacio para la transformación de optimización
        letras_sec_ref = local_buff+sig_multiplo_4(long_sec_ref);
        // Carga del número de letras de la matriz (incluido el *)
        indice_actual += sig_multiplo_4(long_sec_ref);
        e_dma_copy(&word_aux, (void *)buff+indice_actual, sizeof(word_aux));
        e_dma_wait(canal_dma);
        num_letras = (uint8_t)word_aux;
        // Carga del alfabeto traducido (128 posiciones)
        indice_actual += 4;
        e_dma_copy(letras, (void *)buff+indice_actual, sizeof(letras));
        e_dma_wait(canal_dma);
        indice_actual += sizeof(letras); // 128
        // Carga de la matriz. Se leen bytes de más para leer palabras completas.
        matriz = letras_sec_ref+sig_multiplo_4(long_sec_ref); // Lo mismo que local_buff+2*sig_multiplo_4(long_sec_ref);
        e_dma_copy(matriz, (void *)buff+indice_actual, sig_multiplo_4(num_letras*num_letras));
        e_dma_wait(canal_dma);
```

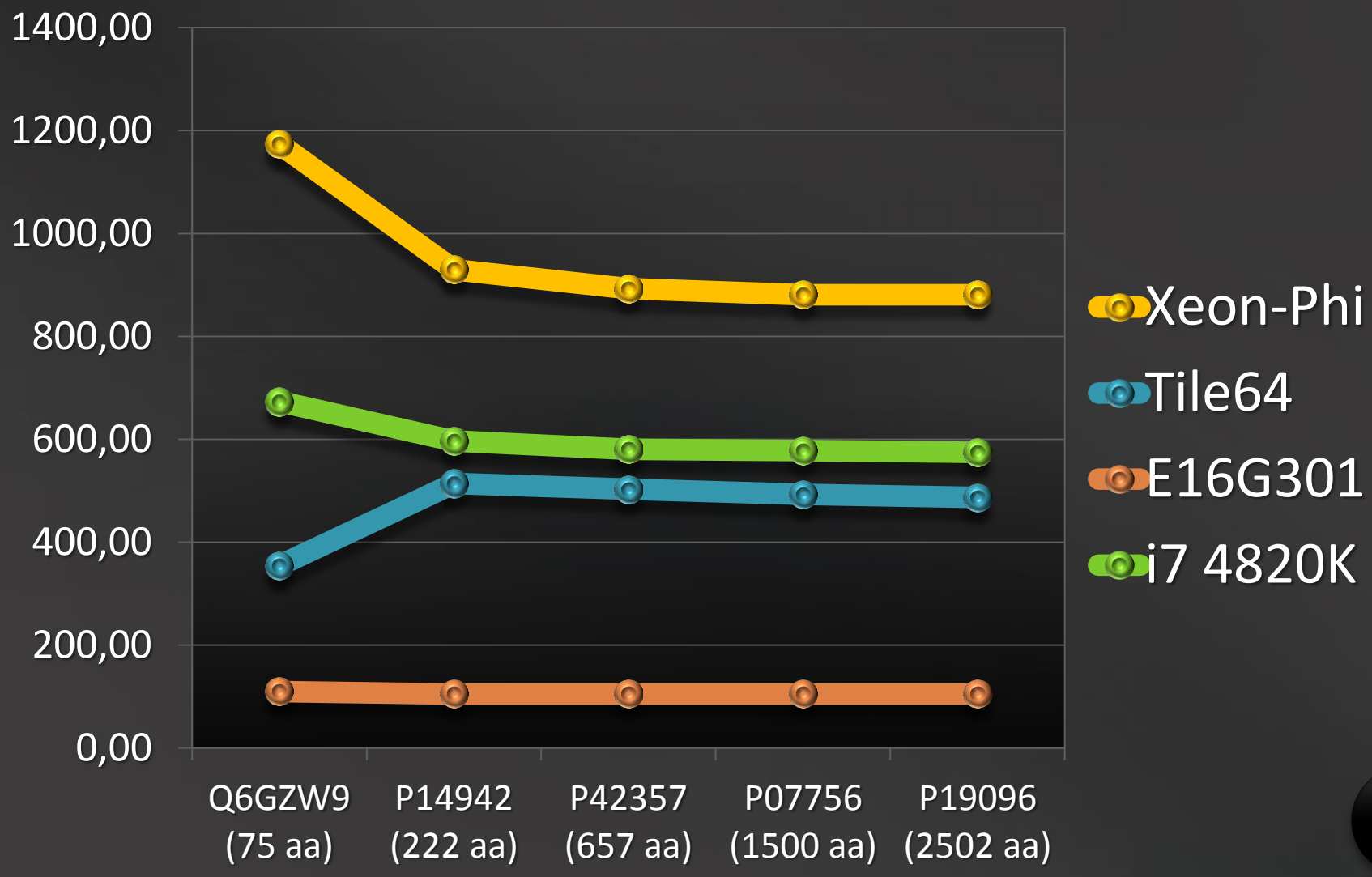
C Ancho de la tabulación: 8 Ln 214, Col 31 INS

Caso práctico

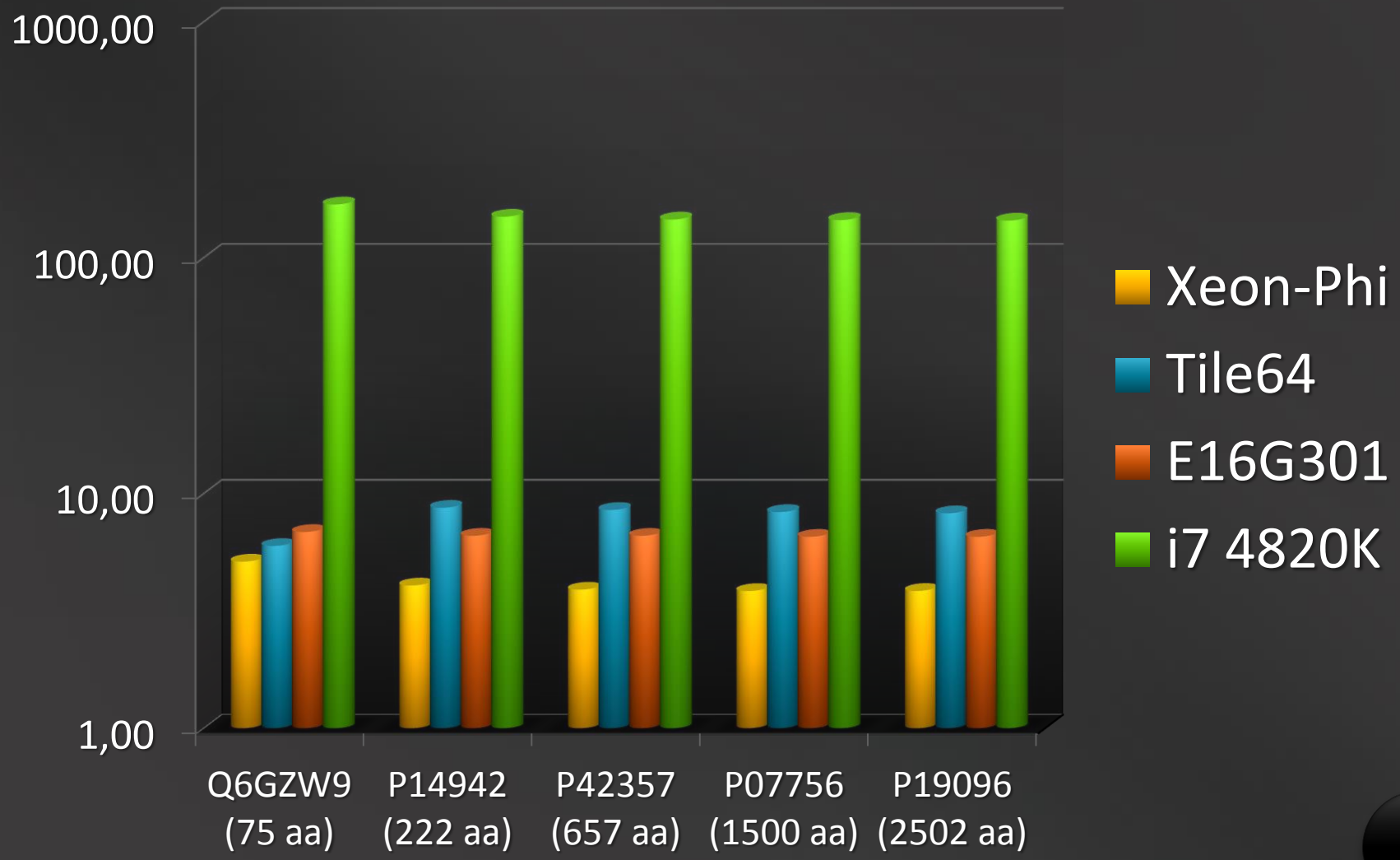
Implementación



Rendimiento absoluto (en MCUPS)



Rendimiento relativo (en MCUPS/hilo)



Conclusiones

- Consumo
- Vectorización
- Capacidad
  - Memoria
  - Soporte de coma flotante
- Rendimiento
- Facilidad de desarrollo
- Soporte y Comunidad
- ¿Futuro?

Preguntas