Computación Paralela 2012 — Universidad Nacional de Córdoba
Práctico 2 - Optimización secuencial — FaMAF

Carlos Bederián, Nicolás Wolovick

Objetivos:

- Implementar algoritmos *cache-aware* y *cache-oblivious*.
- Iniciarse en el uso de *SSE intrinsics*.
- Iniciarse en el uso de un *profiler*.

La siguiente función transpone una matriz bidimensional:

```
void transpose_2d(const float * restrict in, size_t in_stride,
                  size_t width, size_t height,
                  float * restrict out, size_t out_stride) {
    for (size_t y = 0; y < height; ++y) {
        for (size_t x = 0; x < width; ++x) {
            out[y + x * out_stride] = in[x + y * in_stride];
        }
    }
}
```

Ejercicio 1. Discusión: ¿Para qué sirven `in_stride` y `out_stride`?

Ejercicio 2. La función anterior recorre por columnas la matriz destino, generando gran cantidad de cache misses y relecturas. Una forma *cache-aware* de solucionar este problema es aplicar *cache blocking* o *loop tiling*, dividiendo el problema en bloques de tamaño fijo que caben en algún nivel de cache. Implemente este cambio y evalúe la performance para distintos tamaños de bloque.

Ejercicio 3. El enfoque anterior generalmente es óptimo cuando el tamaño de bloque está hecho a la medida del hardware sobre el que corre. Los algoritmos *cache-oblivious* utilizan estrategias que no requieren conocimiento del hardware subyacente. Una forma de hacerlo para este problema es aplicando *divide and conquer*, dividiendo el problema en 4 menores de manera recursiva hasta llegar al caso base de una celda. Implemente este algoritmo y evalúe la performance contra la implementación anterior.

Ejercicio 4. El código más veloz no sólo hace buen uso del sistema de cache, sino también del almacenamiento más veloz y pequeño, los registros. Un procesador de arquitectura x86-64 posee 16 registros XMM de 128 bits, que pueden almacenar 64 celdas. Tome la implementación del ejercicio 2, fije el tamaño de bloque a uno que entre en dichos registros y utilícelos mediante *SSE intrinsics*.

Ejercicio 5. Una implementación robusta del ejercicio anterior utiliza lecturas y escrituras desalineadas. ¿Qué debería darse para poder usar operaciones alineadas? Implemente la detección de estas condiciones y llame a una versión alternativa de la función que haga operaciones alineadas de ser posible. Compare ambas implementaciones en procesadores diversos.

Ejercicio 6. Una forma sencilla de implementar una multiplicación de matrices rápida es trasponer el segundo operando, con lo que la lectura y escritura de las tres matrices intervinientes se hacen de manera secuencial en memoria. Implemente la multiplicación de matrices utilizando este método, y distinga el tiempo que toma la transposición del que toma el producto utilizando un *profiler*.

Ejercicio 7. (Extra) Compare su mejor multiplicación de matrices contra la función `sgemm` de una implementación rápida de BLAS (MKL, GotoBLAS 2) configurada para correr en un core.

Ejercicio 8. (Extra) Tome su implementación más veloz de la transposición y modifíquela para que opere *in-place* sobre la misma matriz origen. Armado de esta función y su mejor multiplicación de matrices, implemente `sgemm` y compare nuevamente contra un BLAS.