High Performance Computing
Departamento de Ingeniería en Informática
LAB1 : SIMD- SSE

1 Objetivo

El objetivo de este laboratorio es conocer, usar y apreciar las características de paralelismo a nivel de instrucción de un procesador moderno. Muchas veces desconocemos por completo las capacidades de cómputo SIMD que los procesadores poseen, y rara vez nos preocupamos de explotarlas.

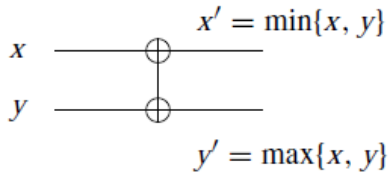
En este laboratorio usted implementará un programa mono-hebreado para ordenamiento de números. Específicamente, usaremos merge bitónico a nivel SIMD para ordenar subsecuencias y luego merge a nivel de proceso para mezclar estas subsecuencias y obtener la secuencia ordenada.

2 Bitonic sorting network

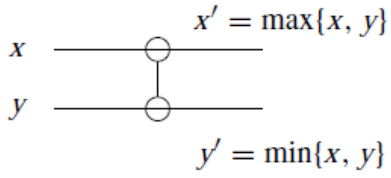
2.1 Sorting network

Una **red de ordenamiento** es un algoritmo de ordenamiento que demora (siempre) el mismo número de pasos usando (siempre) un número fijo de comparaciones. Comúnmente las operaciones son del tipo *comparar-intercambiar*.

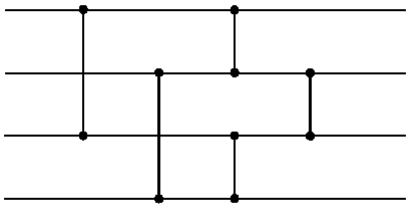
Un **comparador creciente** es un operador con dos entradas x e y y dos salidas x' e y', tal que:



Un **comparador decreciente** es un operador con dos entradas x e y y dos salidas x' e y', tal que:



Las redes de ordenamiento pueden ser implementadas en hardware o software como una secuencias de elementos comparadores. Por ejemplo, la siguiente figura muestra una red de ordenamiento para una secuencia de 4 elementos (en general siempre asumiremos que las secuencias son de largo potencias de dos)



Es importante enfatizar que el número de operaciones y tiempo de ejecución siempre es el mismo, independiente del orden de la secuencia de entrada.

2.2 Bitonic sequences

Una secuencia **monotónicamente creciente** es una secuencia de elementos $[a_0, a_1, \ldots, a_{n-1}]$ tal que $a_i \leq a_j$, si $i < j$. Una secuencia **monotónicamente decreciente** es una secuencia de elementos $[a_0, a_1, \ldots, a_{n-1}]$ tal que $a_i \geq a_j$, si $i < j$.

Una **secuencia bitónica** es una secuencia de elementos $[a_0, a_1, \ldots, a_{n-1}]$ que cumple una de las siguientes propiedades:

1. Existe un k, $0 \leq k \leq n-1$, tal que $[a_0, a_1, \ldots, a_k]$ es monotónicamente creciente y $[a_{k+1}, \ldots, a_{n-1}]$ es monotónicamente decreciente.

2. Existe un shift circular de la secuencia tal que (1) es satisfecho.

Por ejemplo, la secuencia $[8, 10, 15, 2, 1]$ es bitónica pues está compuesta de $[8, 10]$ y $[15, 2, 1]$. La secuencia $[8, 9, 9, 10, 3, 5]$ también es bitónica pues es un shift de $[5, 8, 9, 9, 10, 3]$.

Sea $s = [a_0, a_1, \ldots, a_{n-1}]$ una secuencia bitónica tal que

- $[a_0 \leq a_1 \leq \ldots \leq a_{n/2-1}]$ y
- $[a_{n/2} \geq a_{n/2+1} \geq \ldots \geq a_{n-1}]$

Luego, las secuencias:

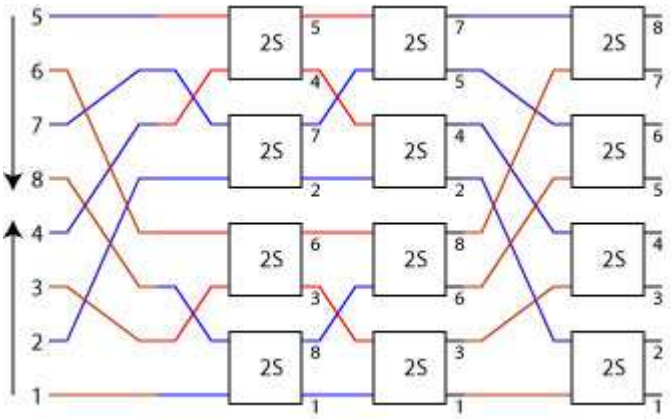
$$\begin{aligned} s_1 &= [\min\{a_0, a_{n/2}\}, \min\{a_1, a_{n/2+1}\}, \ldots, \min\{a_{n/2-1}, a_{n-1}\}] \\ s_2 &= [\max\{a_0, a_{n/2}\}, \max\{a_1, a_{n/2+1}\}, \ldots, \max\{a_{n/2-1}, a_{n-1}\}] \end{aligned}$$

también son bitónicas. **Además, cada elemento de s_1 es menor o igual que cada elemento de s_2!!!**. Ejemplo: $s = [2, 7, 8, 15, 13, 12, 4, 3]$

$$\begin{aligned} s_1 &= [2, 7, 4, 3] \\ s_2 &= [13, 12, 8, 15] \end{aligned}$$

2.3 Bitonic merge network

Una **bitonic merge network (BMN)** es una red compuesta operadores `min()` y `max()` (y shuffles) que toma una secuencia bitónica y entrega la secuencia ordenada. La siguiente es la red Batcher de 8 entradas:



Los dos primera etapas de esta red son:

1. La entrada a la red es una secuencia bitónica, la cual consiste de dos registros cada uno de largo 4.
$$a = [1, 2, 3, 4] \quad b = [8, 7, 6, 5]$$
2. La secuencia bitónica de entrada es reordenada con operadores `min-max`, tal que:
   - El primer elemento de la primera secuencia es comparado con con el primer elemento de la segunda secuencia
   - El tercer elementos de la primera secuencia es comparada con el tercero de la segunda
   - El segundo de la primera con el segundo de la tercera
   - El cuarto de la primera con con el cuarto de la segunda
3. Y se vuelven a reordenar apropiadamente los resultados como indica la figura.

Note que la secuencia final esta compuesta de dos secuencias de cuatro elementos tal que deben mezclarse uno a uno para obtener la secuencia ordenada total. Es decir, en el ejemplo de la figura, las secuencias son:

$$a = [1, 3, 5, 7] \quad b = [2, 4, 6, 8]$$

Luego, la última etapa de la red es un reordenamiento tal que la secuencia final sea:

$$a = [1, 2, 3, 4] \quad b = [5, 6, 7, 8]$$

Lamentablemente, el ejemplo no es muy bueno. Mejor veamos uno distinto.

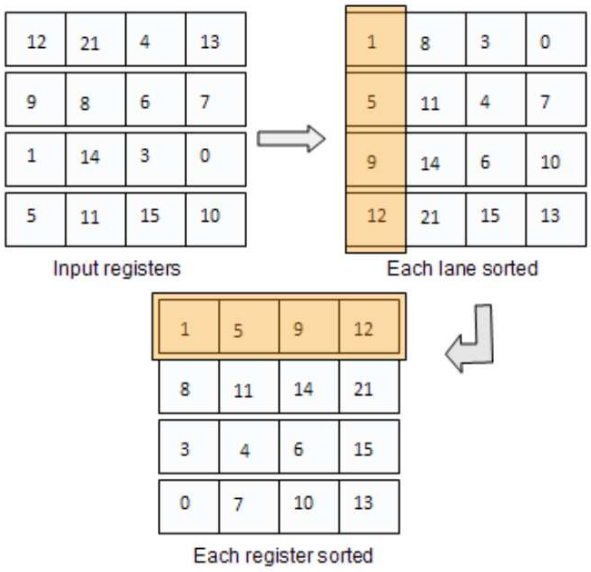
$$a = [5, 20, 25, 30] \quad b = [18, 9, 7, 2]$$

2.4 Ordenamiento in-register

Para alimentar la red anterior necesitamos dos secuencias ordenadas de menor a mayor, y revertir el orden de la segunda. Asi obtenemos la secuencia bitónica. Lo que falta es cómo obtener secuencias ordenadas. Para esto, leemos 16 números de la secuencia original en 4 registros. Por ejemplo, la secuencia

$$s = [12, 21, 4, 13, 9, 8, 6, 7, 1, 14, 3, 0, 5, 11, 15, 10]$$

al ser leidas quedaría almacenada como lo muestra la siguiente figura:



Para obtener secuencias ordenadas, se realiza el siguiente procedimiento:

1. Cada posición de los cuatro registros se considera una pista, es decir a_i, b_i, c_i, d_i es la pista i.
2. Cada pista se ordena usando una red de operadores `min-max` para cuatro elementos de entrada.
3. Se aplica una red de shuffles para *trasponer* los elementos, tal que los elementos quedan almacenados como lo muestra la última matriz.

Al final de este procedimiento tenemos cuatro secuencias ordenadas de cuatro elementos cada una. Finalmente, se utiliza la BMN (dos veces) para producir dos secuencia ordenadas de largo 8. Es decir, para el ejemplo anterior, tendriamos:

$$S_1 = [1, 5, 8, 9, 11, 12, 14, 21] \quad S_2 = [0, 3, 4, 6, 7, 10, 13, 15]$$

2.5 Merge SIMD de secuencias

Para mezclar dos secuencias ordenadas de 8 elementos cada una, empleamos la misma lógica que el algoritmo merge, pero usando la BMN, en cada iteración.

1. Se cargan los primeros 4 elementos de las dos secuencias y se alimentan a la BMN. El resultado son dos registros O_1 y O_2, tal que $[O_1, O_2]$ es una secuencia ordenada.

2. Se extrae O_1 y se almacena en la secuencia de salida.

3. O_2 pasa a ser O_1 (en realidad esto no es necesario si se programa bien)

4. Para cargar O_2, se compara los dos primeros elementos, de las secuencias que quedan. O_2 se carga con la secuencia que tenga el primer elemento menor

5. Volvemos al paso 1.

El procedimiento anterior requiere la aplicación de 4 BMN y da como resultado una secuencia ordenada de 16 elementos.

3 Multiway Merge Sort (MWMS)

Suponga que el largo de la secuencia a ordenar es $N = 2^k$, para algún k. El procedimiento anterior se repite cada 16 elementos de esta lista dando como resultado $L = \frac{N}{16}$ listas ordenadas. En principio, sería posible reutilizar la BMN para seguir mezclando listas de 16 elementos, pero por ahora saldremos de las unidades SIMD para realizar un MWMS de L listas ordenadas, como se indica a continuación:

1. Se encuentra el número menor entre las listas. Para esto sólo basta comparar los primeros elementos de cada lista, pues todas las listas están ordenadas.

2. Se saca el elmentos menor de su lista y se mueve a la lista final.

3. Volvemos al paso 1, hasta que todas las listas queden vacías.

Note que durante el MWMS es probable que las listas de entrada se vacíen y por lo tanto, ya no sería necesario incluirlas en la búqueda del menor.

4 Putting it all together

El programa que ustedes deben implementar consiste de dos partes importantes, una SIMD y otra no SIMD.

SIMD : Esta fase tiene como objetivo producir secuencias ordenadas de 16 elementos, usando el BMN, shuffles y una red `min-max` para ordenar 4 elementos. Idealmente, los 16 elementos se llevan al procesador y no vuelven a la memoria hasta que no están ordenados. Esta fase se puede separar en las siguientes subfases:

1. Llevar 16 elementos al procesador

2. Obtener 4 secuencias ordenadas de menor a mayor, usando la red `min-max` y los shuffles para realizar el traspose

3. Obtener dos secuencias ordenadas de largo 8 usando la BMN

4. Obtener una secuencia ordenada de largo 16 usando Merge SIMD

5. Almacenar la secuencia ordenada en memoria

No SIMD : Esta fase mezcla todas las lista ordenadas de 16 elementos en una sola lista final ordenada de N elementos. Se utiliza sólo una hebra para realizar esta operación, y un algoritmo simple para encontrar el mínimo entre L elementos.

5 Los programas y la lista

Usted debe implementar este programa en lenguaje C, sobre sistema operativo Linux. El programa debe ejecutarse de la siguiente forma:

```
$ ./simdsort -i desordenada.raw -o ordenada.raw -N num_elementos -d debug
```

donde las opciones indican lo siguiente:

- `-i`: archivo binario con la lista de entrada desordenados
- `-o`: archivo binario de salida con la lista ordenada
- `-N`: largo de la lista
- `-d`: Cuando debug=0, no se debe imprimir ningún mensaje por stdout. Cuando debug=1, se debe imprimir la secuencia final por stdout, un elemento por línea.

Para este programa, usaremos siempre listas con números enteros (32 bits) y de largo potencia de dos. Los números en los archivos de entrada y de salida deben estar almacenados en formato binario. Para leer y escribir en binario utilice los llamados al sistema `read()` y `write()`. Esto permitirá usar el comando `diff` para chequear qua la salida es correcta.

Sin embargo, para desarrollar usted puede utilizar `printf()` u otras funciones para imprimir resultados parciales, pero todas estas impresiones se deben deshabilitar en el entregable. Sólo se permite imprimir según el nivel debug especificado, nada más.

6 Entregables

Tarree, comprima y envíe a `fernando.rannou@usach.cl` al menos los siguientes archivos:

1. `Makefile`: archivo para make que compila los programas
2. `simdsort.c`: archivo con el código. Puede incluir otros archivos fuentes.
3. `simdsort.pdf`: informe breve de su solución, a nivel de diagramas y algoritmos.

Fecha de entrega: domingo 8 de noviembre antes de las 24:00 hrs.