Scientific High-Performance Computing

Obryl Donald Houaghon Nankap[1] and Weimbapou Siewe[2]

[1]obryl.donald.houaghon.nankap@tu-clausthal.de
[2]weimbapou.siewe@tu-clausthal.de

24 November 2021

Sheet 1

Exercice 3 : Instruction Level Parallelism

Instruction Level Parallelism (ILP) is a measure of how many operations of a program can be processed in parallel. In this task, we examine how various data and control flow dependencies can affect ILP

```cpp
// A struct for the elements to be inserted.
typedef struct Element {
  int value;
  struct Element* next;
} Element;

// The array of items.
Element my_elements[N_ELEMENTS];
// 1024 buckets, pointers in each bucket are initialized to nullptr.
Element* bucket[1024];

for (size_t i = 0; i < N_ELEMENTS; ++i) {
  Element* ptr_curr;
  Element** ptr_update;
  int hash_index;

  // Find location where the new element is to be inserted.
  hash_index = my_elements[i].value & 1023;
  ptr_update = &bucket[hash_index];
  ptr_curr   = bucket[hash_index];

  // Find place in chain to insert the new element.
  while (ptr_curr && ptr_curr->value <= my_elements[i].value) {
    ptr_update = &ptr_curr->next;
    ptr_curr   = ptr_curr->next;
  }

  // Update pointers to insert the new element into chain.
  my_elements[i].next = *ptr_update;
  *ptr_update = &my_elements[i];
}
```

Figure 1: The *Incomplete Hash source code*

Only the answer are provided

1. Which dependency exists between the lines 18 and 19/20? : Data true dependancy between 21 and 22, and 23 since the hashIndex is needed to produce the hash address.

2. Which dependency exists between the lines 18 and the same line 18 for two different elements with the same hash value? : data output dependences, because both share and write on the same address at the time they get executed

3. Which dependency exists between the lines 30 and 19?: data antidepedance which is here valid because the instructions 19 write address which is at line 30 used to store the current pointer update of element.

1. What is the only remaining dependency between the iterations (consider the variable i in line 12)?: data true dependency is the only dependency wich in each iterration modify the position of each value in the table myElement and the hashIndex in the bucket pointer value and address.

2. Rewrite the for-loop in such a way that the dependency between the iterations is reduced (tip: implement instructions for i and i+1 in one loop).

```
for (size_t i = 0; i < N_ELEMENTS && ptr_curr && ptr_curr->value
<= my_elements[i].value; ++i) {
Element* ptr_curr;
Element** ptr_update;
int hash_index;

// Find location where the new element is to be inserted.
hash_index = my_elements[i].value & 1023;
ptr_update = &bucket[hash_index];
ptr_curr = bucket[hash_index];

// Find place in chain to insert the new element.
ptr_update = &ptr_curr->next;
ptr_curr = ptr_curr->next;


// Update pointers to insert the new element into chain.
my_elements[i].next = *ptr_update;
*ptr_update = &my_elements[i];
}
```

3. What influence does your optimisation from (b) have on the ILP? : It save some time and make the code run faster since the loop is now one way and do not test any condition when it enter the loop. Since here the data on i is updated in one place at the begining of the loop.

Exercice 2 : Processor Properties

1. Determine some basic properties of your computer's processor:

   - Manufacturer : Intel, Core i5 9300h
   - Number of logical processor core: 4, physical processor core: 4
   - Frequence of the a processor core : @2.40Ghz
   - Maximum CPU frequency: 4.10 Ghz, yes it support dynamic scalling
   - Cache size: L3 cache: 8Mo, L2 cache: 1Mo, L1 cache: 256k.
   - Instruction and set extentions: MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, Intel 64, NX, VMX, AES, AVX, AVX2, FMA3.

2. The *STREAM* benchmark

   - Determine the memory bandwidth depending on the size of the felds A and B using the COPY benchmark : Best Rate MB/s = 7853.5, Avg time =0.021498, Min time = 0.020373, Max time = 0.022869. Array size = 10000000 (elements), Offset = 0 (elements) Memory per array = 76.3 MiB (= 0.1 GiB). Total memory required = 228.9 MiB (= 0.2 GiB). Each kernel will be executed 10 times.
   - What can you learn about the properties of the caches or memory from this? It is clear that the cache memory is way faster that the main memory. And data that are loaded into it can be faster reuse by a program. Moreover the access time is also very low.
   - Which benchmark would you choose to test the maximum computing speed? What results do you get? I will choose the TRIAD because it is more complex and use 4 kernels. This result give me an overview of how my computer will act in difficult situation. The result are : Best Rate MB/s = 10495.5, Avg time = 0.023850, Min time = 0.022867, Max time = 0.025931.

Exercice 3 :Measuring MFLOPS

1. MFLOPS mesusing of the matrix code : The MFLOP begin to drop when the matrix the elements become bigger and bigger. The reason here is the usage of the main memory to process and transfer the data. It is clear that the Cache usage help the program to run very fast. But the main memory is not that faster to run the same but it is bigger to contain all the necessary data for the program. for example here

```
        for (int i = 0; i < matriceLenght; i++)
    for (int j = 0; j < matriceLenght; j++)
      for (int n = 0; n < matriceLenght; n++) // when we move this instruction at the b
         matrixResult[i][j] = matrixResult[i][j] + matrix[i][n] * matrix2[n][j];  //cod
```

The same process is made on the container itself by making it accessible in one row in the memory. A contiguous data in the memory is more faster and efficient.

```
int** matrix = new int* [matriceLenght]; // pointer to allocate all the matric
```

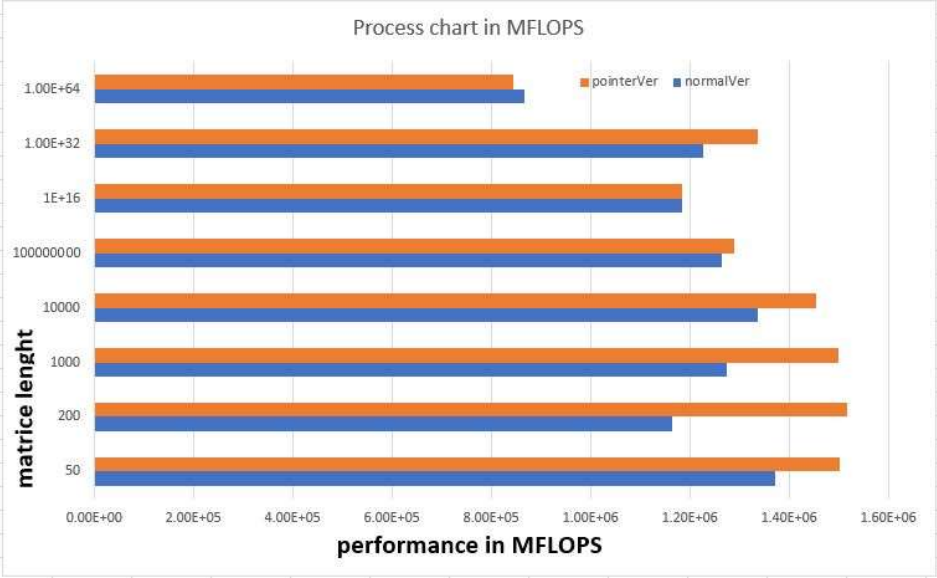

Figure 2: The *MFLOPS chart, higher is better*

2. The loop reordering help the program to run very fast. For example the reading of row first instead of column in the calculation loop help the program to process faster.

3. The Gauss seidel investigation : The chart tell us that when the cache is full the main memory is again used the face the situation and the result of less velocity to access memory affect the two version of the code. The loop reordering also have a impact of on execution. Move the the instruction to read row first and columns afterwards can be very useful in certain case.

4. Can reordering the loops improve the cache utilization in this case? Yes loop reordering can help to make data fit in the cache before processing. By making more nested loop to have access on the data in small blocks and process it with others. for example by using more nested loop.

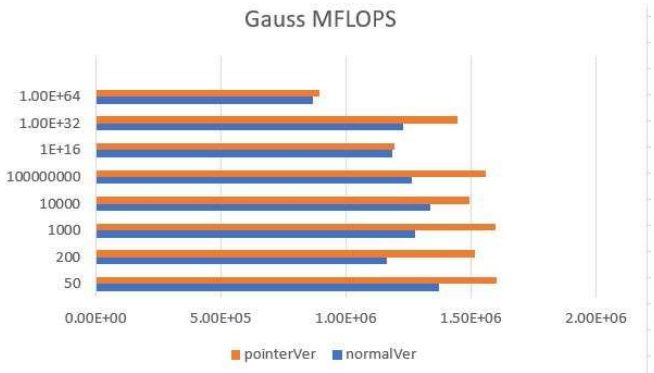

Figure 3: The *MFLOPS chart, higher is better*