RAPPORT TP CC OPENMP

Résultats préliminaires : sans optimisations :

```
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ g++ -g -fopenmp RiemannSiegel.cpp -O0 -o Riemann
Siegel
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegel 10 10000 100
I estimate I will find 10141.965 zeros
I found 10142 Zeros in 4.338 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegel 10 100000 100
I estimate I will find 138067.558 zeros
I found 138069 Zeros in 79.266 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$
```

1- Profiling avec vtune : détermination des hotspots :

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
powf64	libm.so.6	1.868s	42.7%
powf64x	libm.so.6	0.826s	18.9%
func@0x87564	libm.so.6	0.436s	10.0%
Z	RiemannSiegel	0.300s	6.9%
__fmod_finite	libm.so.6	0.272s	6.2%
[Others]	N/A*	0.668s	15.3%

*N/A is applied to non-summable metrics.

OPTIMISATION SINGLE CODE :

1- on a essayé d'éliminer les fonctions qui nous coûtent cher :

1-1 élimination de pow j'ai eu trois idee :

1- construire une fonction powf :

```
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 1000 100
I estimate I will find 647.616 zeros
I found 649 Zeros in 0.651 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 10000 100
I estimate I will find 10141.965 zeros
I found 10142 Zeros in 7.559 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 100000 100
I estimate I will find 138067.558 zeros
I found 138069 Zeros in 113.034 seconds
```

2- dans C : stocker les puissances de z dans un tableau tab puis les recuperer :

```
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 1000 100
I estimate I will find 647.616 zeros
I found 649 Zeros in 0.304 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 10000 100
I estimate I will find 10141.965 zeros
I found 10142 Zeros in 3.964 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 100000 100
I estimate I will find 138067.558 zeros
I found 138069 Zeros in 76.718 seconds
```

3- transformer les pows dans C en multiplication elimentaires, pow(z,2) = z*z : J'ai choisi celle la pour continuer avec.

```
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 1000 100
I estimate I will find 647.616 zeros
I found 649 Zeros in 0.242 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 10000 100
I estimate I will find 10141.965 zeros
I found 10142 Zeros in 3.378 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 100000 100
I estimate I will find 138067.558 zeros
I found 138069 Zeros in 70.537 seconds
```

4- eliminer les pow dans theta :

```
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 1000 100
I estimate I will find 647.616 zeros
I found 649 Zeros in 0.162 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 10000 100
I estimate I will find 10141.965 zeros
I found 10142 Zeros in 2.589 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 100000 100
I estimate I will find 138067.558 zeros
I found 138069 Zeros in 62.602 seconds
```

5- eliminer fmod : Ca se voit que ca optimise quand on passe a 100 000

```
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 1000 100
I estimate I will find 647.616 zeros
I found 649 Zeros in 0.161 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 10000 100
I estimate I will find 10141.965 zeros
I found 10142 Zeros in 2.573 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 100000 100
I estimate I will find 138067.558 zeros
I found 138069 Zeros in 56.727 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$
```

6- Dans la main, le deuxième argument de Z est fixe a 4, si on le trace dans Z, on voit que c est ce que l'on passe a C comme deuxieme argument, et pour ca je vais essayer a deconstruire la boucle for et voir si ca va nous donnes des résultats :

```
    double R  = 0.0;

    double ha = 2.0*pi/t;
    double haha = 2.0*p-1.0;
    double square = sqrt((double) ha);
    double poweroftwo = ha*ha;

        R = C(0,haha) + C(1,haha) * square + C(3,haha) * poweroftwo * square + C(4,haha) *
poweroftwo;
```

```
I estimate I will find 647.616 zeros
I found 649 Zeros in 0.132 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 10000 100
I estimate I will find 10141.965 zeros
I found 10142 Zeros in 2.284 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 100000 100
I estimate I will find 138067.558 zeros
I found 138069 Zeros in 53.796 seconds
```

Ça aide !!

OPTIMIZATION FLAGS :

En compilant avec g++ -g -fopenmp RiemannSiegel1copy.cpp -O0 -o RiemannSiegelcopy ,

Vtune nous indique un score faible de vectorization et de l exploitation du microarchitecture, alors j'ai décidé de mettre des "inline" pour indiquer au compilateur qu'il faut essayer de vectoriser certaines parties de notre code, mais ca ne marchait pas du tout.
Cela m'a incité à essayer de compiler avec les flags O2 O3, sauf que cela marchait pas et j'ai eu une erreur un peu bizarre :

```
RiemannSiegel.cpp: In function 'void test_fileof_zeros(const char*)':
RiemannSiegel.cpp:344:8: warning: ignoring return value of 'char* fgets(char*, int, FILE*)', declared with attrib
ute warn_unused_result [-Wunused-result]
  344 |    fgets(line,1000,fi);
      |    ~~~~~^~~~~~~~~~~~~~
```

J'ai pu contourner ca par deux méthodes, soit en commentant la fonction entièrement, soit par la déclaration d'une variable globale comme suit :

```
char line[1024];
void test_fileof_zeros(const char *fname)
{
    FILE *fi=fopen(fname,"r");
    assert(fi!=NULL);
    for(;;){
        double t,RS;
        char *glob ATTRIBUTE_UNUSED;
        glob = fgets(line, 1000, fi); e
        if(feof(fi)) break;
        sscanf(line,"%lf",&t);
        RS=Z(t,4);
        printf(" %30.20lf %30.20lf   %s",t,RS,line);

    }
    fclose(fi);
}
```

BENEFITS OF COMPILING WITH O2 O3 :

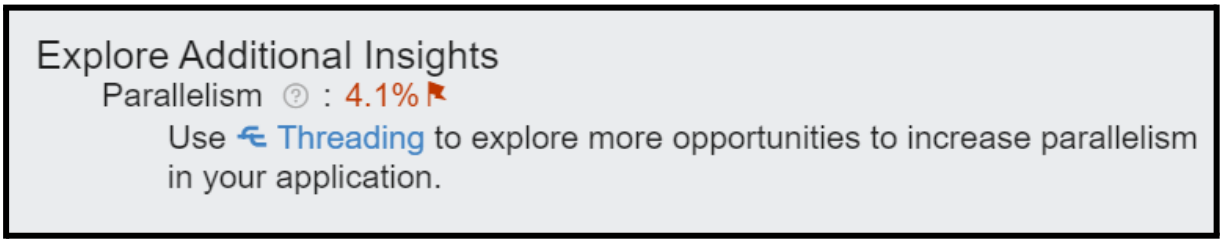
En faisant ca, on constate une amélioration immédiatement :

```
u216367@s001-n058:~/oneAPI-samples/Tools/Advisor/projet_advisor$ g++ -g -fopenmp RiemannSiegel1copy.cpp -O3 -o Ri
emannSiegelcopy
u216367@s001-n058:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 1000 100
I estimate I will find 647.616 zeros
I found 649 Zeros in 0.081 seconds
u216367@s001-n058:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 10000 100
I estimate I will find 10141.965 zeros
I found 10142 Zeros in 1.383 seconds
u216367@s001-n058:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 100000 100
I estimate I will find 138067.558 zeros
I found 138069 Zeros in 32.395 seconds
```

OPTIMISATION AVEC OPENMP :

- **Le rapport de synthèse de VTune indique une faible exploitation du parallélisme : j'ai généré ce rapport sans avoir compilé avec les options O2 ou O3, puis je l'ai refait avec O2/O3, mais le résultat reste le même.**

-



FIRST IDEA :

Ma première idée c'était d'utiliser un #pragma omp parallel for sur la boucle dans Z :

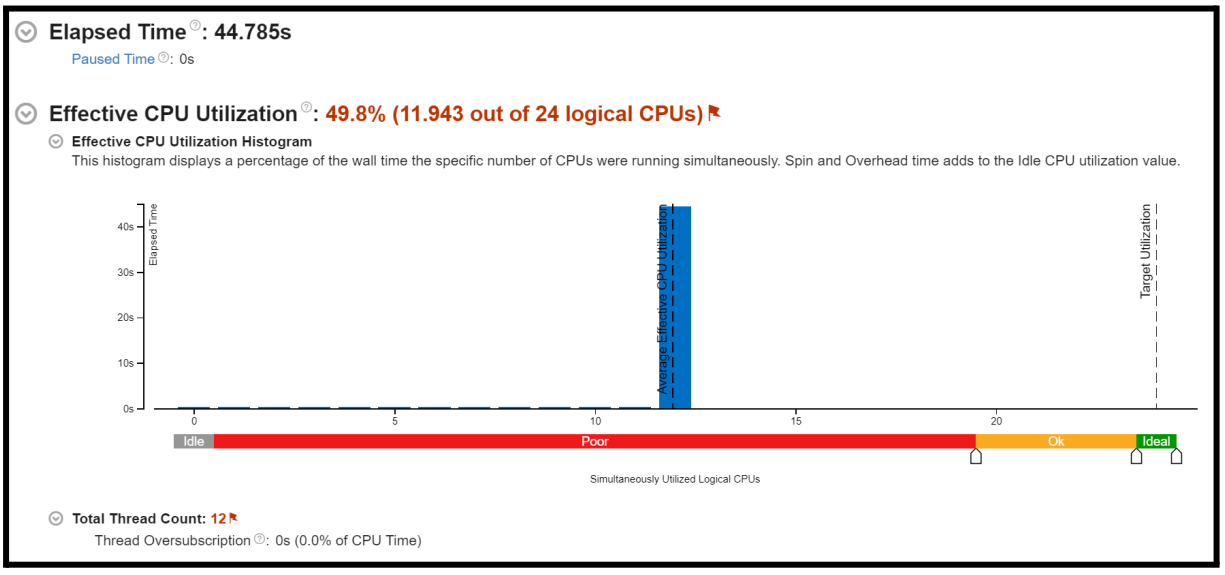
`int N = sqrt(t/(2.0 * pi));`		
`p = sqrt(t/(2.0 * pi)) - N;`	0.0%	0.012s
`double tt = theta(t);`	0.1%	0.032s
`double ZZ = 0.0;`	0.0%	0.008s
`for (int j=1;j <= N;j++) {`		
`ZZ += 1.0/sqrt((double) j ) * cos(myfmod(tt -t*log((double) j),2.0*pi));`	0.9%	0.460s
	14.0%	7.548s
`}`		
`ZZ = 2.0 * ZZ;`		
`double R  = 0.0;`	0.0%	0.016s
	0.0%	0.012s
`double ha = 2.0*pi/t;`		
`double haha = 2.0*p-1.0;`		
`double square = sqrt((double) ha);`	0.0%	0.008s
`double poweroftwo = ha*ha;`		

```
#pragma omp parallel for reduction(+:ZZ)
for (int j=1;j <= N;j++) {
    double tmp = 1.0/sqrt((double) j ) * cos(myfmod(tt -t*log((double) j),2.0*pi));
    ZZ+=tmp;
}
```

il s'avère que ca n'optimise que quand le nombre de 0 qu'il cherche est grand :

```
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 1000 100
I estimate I will find 647.616 zeros
I found 649 Zeros in 0.440 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 10000 100
I estimate I will find 10141.965 zeros
I found 10142 Zeros in 4.348 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 100000 100
I estimate I will find 138067.558 zeros
I found 138069 Zeros in 46.528 seconds
```

on fait une exploitation de 50% des CORES disponible



Setting number of threads to 8 does optimize our application :

```
I estimate I will find 647.616 zeros
I found 649 Zeros in 0.261 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 10000 100
I estimate I will find 10141.965 zeros
I found 10142 Zeros in 2.814 seconds
u216367@s001-n002:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 100000 100
I estimate I will find 138067.558 zeros
I found 138069 Zeros in 35.746 seconds
```

BUT, by doing that we limit ourselves to 8 threads instead of 24 available

SECOND IDEA :

Ma deuxième idée était de créer l'équipe de threads dans la boucle principale plutôt que dans Z, étant donné que nous bouclons sur Z(t,4), nous faisons donc créer une équipe de threads à chaque variation de t, et ça c'est coûteux.

Mais pour faire ca, on devait changer l'itérateur t de la boucle principale (parallel omp for ne s'applique que sur un itérateur de type int), et faire une manipulation sur LOWER UPPER, pour qu'on puisse avoir le même nombre d'itération, ainsi on devait restaurer le pas STEP = 1.0/SAMP

```
    for (int t=0;t<NUMSAMPLES;t+=1){
        double h = LOWER + (double)t/SAMP ; // restorer le pas 1.0/SAMP t le t LOWER UPPER
        zout=Z(t,4);
        if(h>LOWER){
            if(  ((zout<0.0)and(prev>0.0))
                or((zout>0.0)and(prev<0.0))){
                //printf("%20.6lf  %20.12lf %20.12lf\n",t,prev,zout);
                count++;
            }
        }

        prev=zout;
    }
```

J'ai fais un #pragma omp parallel for reduction(+:count) sur la boucle principale, mais le nombre de 0 calculé était différent à chaque exécution, malgré que le temps d'exécution était bon
Et je pense que c'est dû à l' instruction prev=zout, non synchronisée entre les different thread

Et comme solution j'ai éliminé prev=zout, et remplacé prev et zout par Z(h,4),Z(h-STEP,4) :

```
#pragma omp parallel for reduction(+:count) schedule(runtime) shared(LOWER,STEP)
for (int t = 0; t<NUMSAMPLES;t++) {
    double h = LOWER + (double) t*STEP;  // restorer notre t double pour le passe
    zout = Z(h,4);
    if (h > LOWER) {
        if(   ((Z(h,4)<0.0)and(Z(h-STEP,4)>0.0))
            or((Z(h,4)>0.0)and(Z(h-STEP,4)<0.0))){
            //printf("%20.6lf  %20.12lf %20.12lf\n",t,prev,zout);
               #pragma omp critical
            count++;
        }
    }
       #pragma omp critical
       #pragma omp task if()
       prev = zout;
}
```

Et ça a bien marché, maintenant j'ai le bon nombre de zéros, et une performance améliorée.

```
u216367@s001-n058:~/oneAPI-samples/Tools/Advisor/projet_advisor$ g++ -g -fopenmp RiemannSiegel1copy.cpp -O3 -o Ri
emannSiegelcopy
u216367@s001-n058:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 1000 100
I estimate I will find 647.616 zeros
I found 649 Zeros in 0.056 seconds
u216367@s001-n058:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 10000 100
I estimate I will find 10141.965 zeros
I found 10142 Zeros in 0.741 seconds
u216367@s001-n058:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 100000 100
I estimate I will find 138067.558 zeros
I found 138069 Zeros in 16.660 seconds
```

J'ai essayé avec pragma omp parallel au lieu de parallel for, en prenant chaque itération comme task, mais ça n'a pas donné un meilleur temps d'exécution, plus le nombre de zéros n'est pas correct.

```
  #pragma omp parallel
  {
  #pragma omp single nowait
  for (int t = 0; t<NUMSAMPLES;t++) {
      #pragma omp task
      {
      double h = LOWER + (double) t*STEP;  // restorer notre t double pour le passer a Z()
      zout = Z(h,4);
      if (h > LOWER) {
          if(   ((Z(h,4)<0.0)and(Z(h-STEP,4)>0.0))
              or((Z(h,4)>0.0)and(Z(h-STEP,4)<0.0))){
              //printf("%20.6lf  %20.12lf %20.12lf\n",t,prev,zout);
/                #pragma omp critical
              count++;
          }
      }
      }
/         #pragma omp critical
/         #pragma omp task if()
/         prev = zout;
  }
  }
  #pragma omp taskwait
```

```
u216367@s001-n058:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 1000 100
I estimate I will find 647.616 zeros
I found 660 Zeros in 0.260 seconds
u216367@s001-n058:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 10000 100
I estimate I will find 10141.965 zeros
I found 10236 Zeros in 2.429 seconds
u216367@s001-n058:~/oneAPI-samples/Tools/Advisor/projet_advisor$ ./RiemannSiegelcopy 10 100000 100
I estimate I will find 138067.558 zeros
I found 141278 Zeros in 23.331 seconds
```