Arjun Ray

Facility for Rare Isotope Beams, Michigan State University, East Lansing, MI 48824 USA

2025

This material is based upon work supported by the U.S. Department of Energy Office of Science under Cooperative Agreement DE-SC0000661, the State of Michigan and Michigan State University. Michigan State University operates FRIB as a DOE Office of Science National User Facility in support of the mission of the Office of Nuclear Physics.

BOTTLENECK POINTS AND POSSIBLE OPTIMIZATIONS METHODS

SOME BOTTLENECK POINTS:

Function digam in e_Tceis

```
if(z == c1)  {  psi = -c1*gama;  return;}


if(z.real() < 0)  psi = diser(c1-z) - PI*cos(PI*z)/sin(PI*z);
else              psi = diser(z);
}
```

Function chypser in e_Tceis

```
848                 while(1)
849                     {
850     3.42 min        cfac=((caa*cbb)/ccc)*cfac;
851     1.42 min        cfac=cfac*x/double(n);
852      31.16 s        cf=ctemp+cfac;
853      48.81 s        if(cf == ctemp) break;
854
855                     ctemp=cf;
856   920.00 ms         n=n+1;
857      34.51 s        caa=caa+c1;
858      31.05 s        cbb=cbb+c1;
859      28.84 s        ccc=ccc+c1;
860                     }
```

BOTTLENECK POINTS AFTER PRESSING THE "CONTINUE" BUTTON

```
int f_numPP(int NCO)
{
int numPP=NUM3[NCO+1];
return numPP;
}
```

In e_F4

```
void CalcOlegSum(double *V)
{
static int counter=0;

      for(int i=0; i<8; i++) sumM[i]=0;

      for(int I=1;   I<=63;   I++) sumM[0] += V[I];
      for(int I=64;  I<=66;   I++) sumM[1] += V[I];
      for(int I=67;  I<=73;   I++) sumM[2] += V[I];
      for(int I=74;  I<=84;   I++) sumM[3] += V[I];
      for(int I=85;  I<=293;  I++) sumM[4] += V[I];
      for(int I=294; I<=326;  I++) sumM[5] += V[I];
      for(int I=327; I<=1283; I++) sumM[6] += V[I];

      for(int i=0; i<7; i++)
                  if(i!=4)
                        sumM[7] += (sumM[i]-1.);

counter++;

}
```

In e_ETACHA

POSSIBLE SOLUTIONS:

-First step will be to optimize numerical calculations by reducing redundant calculations(reducing iterations and more efficient methods of calculation, using temporary variables)

-Parallelize the computation of loops in heavy functions.

-Approximations for Large Iterations

Bottleneck Solutions #1

List of B-N	N~ of Bottleneck w/ line number
e_Tceis	5 from lines 850-859
e_Tceis	Function cGamLn lines 1242-1269
e_Tceis	Function diser lines 1027-1033
e_ETACHA	Function CalcOlegSum lines 1317-1331
e_F4	Function f_numPP line 673
	Function f_numP line 636
	Function f_KK line line 534
e_Tceis	Function digam all lines 879-884

```
while(1)
        {
          qDebug()<<n<<cf.real()<<cf.imag()<<ctemp.real()<<ctemp.imag();
        cfac=((caa*cbb)/ccc)*cfac;
        cfac=cfac*x/double(n);
        cf=ctemp+cfac;
        if(cf == ctemp) break;

        ctemp=cf;
        n=n+1;
        caa=caa+c1;
        cbb=cbb+c1;
        ccc=ccc+c1;
        }
}
```

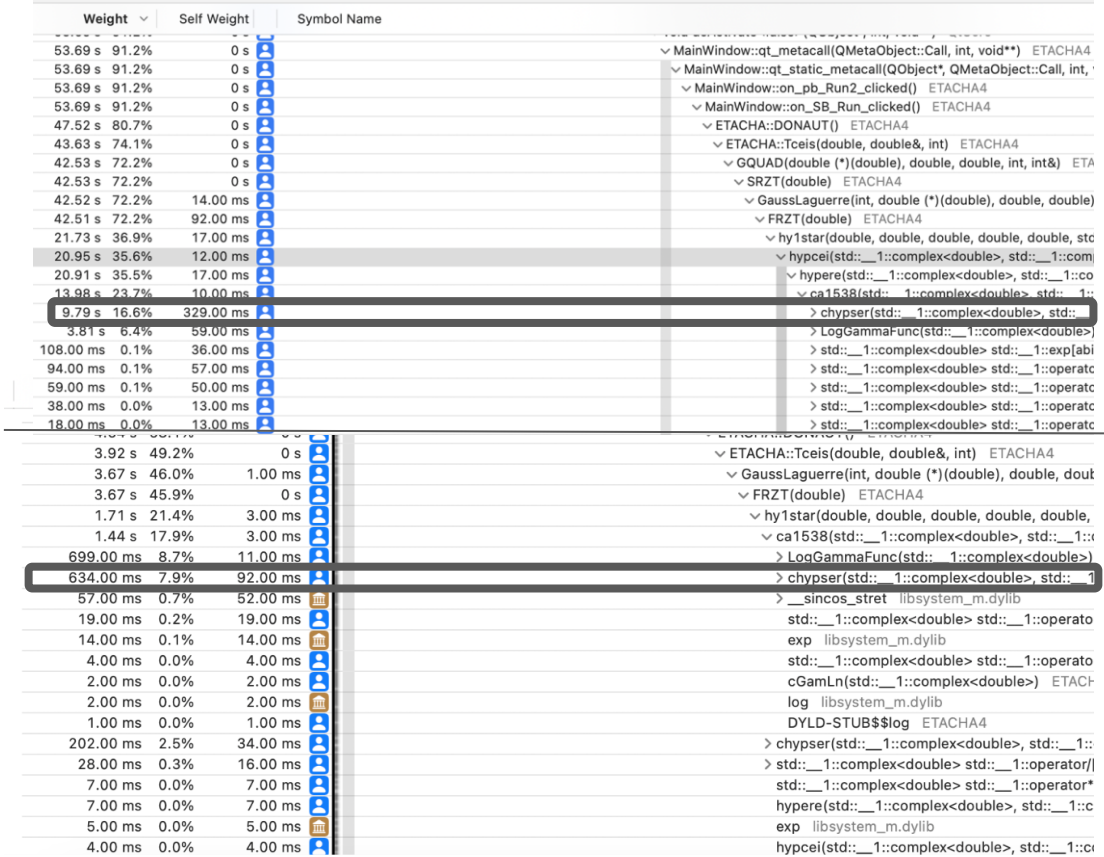
METHODS THAT WE WILL USE TO OPTIMIZE-

-Precompute (caa * cbb / ccc) to avoid redundant calculations.

-Use `abs(cf - ctemp) < EPSILON` instead of direct equality for convergence.

-Parallelize with `#pragma omp parallel` for faster execution.

Result #1



-Made three key improvements to make the function run faster and more reliably:

- Instead of doing the same calculation over and over, it now computes it once and reuses it.
- Instead of looking for an exact match, it checks if the difference is small enough to safely stop.
- Organized the loop better so the function doesn't waste effort.

-Results of optimisation

- Overall function runtime cut by almost 60%
- No longer the heaviest function in e_Tceis

#2

List of B-N	N~ of Bottleneck w/ line number
e_Tceis	5 from lines 850-859
e_Tceis	Function cGamLn lines 1242-1269
e_Tceis	Function diser lines 1027-1033
e_ETACHA	Function CalcOlegSum lines 1317-1331
e_F4	Function f_numPP line 673
	Function f_numP line 636
	Function f_KK line line 534
e_Tceis	Function digam all lines 879-884

```
complex<double>  cGamLn(complex<double> cz)
{
      double rcof[7] = {0,
                        76.18009172947146e0,-86.50532032941677e0,
                        24.01409824083091e0,-1.231739572450155e0,
                        0.1208650973866179e-2,-0.5395239384953e-5};

      double stp=2.5066282746310005;

      if(cz.real() <= 0.) return c0; //// 'real[z] <= 0';

      complex<double> cx(cz);
      complex<double> cy(cx);
      complex<double> ctmp(cx+5.5);
      complex<double> cser(1.000000000190015,0);

      ctmp=(cx+0.5)*log(ctmp)-ctmp;

      for(int j=1; j<=6; j++)
                  {
                  cy   = cy+1.;
                  cser = cser+rcof[j]/cy;
                  }

complex<double> canswer=ctmp+log(stp*cser/cx);

return canswer;
}
```

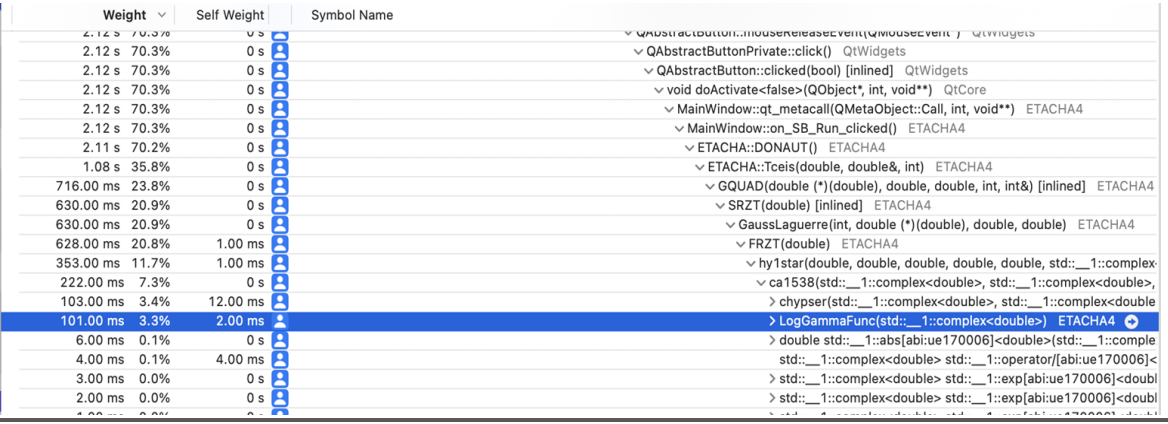
METHODS THAT WE WILL USE TO OPTIMIZE-

-Use of constexpr for Constants

-Avoided unnecessary pre computations in loop

-Use fewer temporary variables in return statement

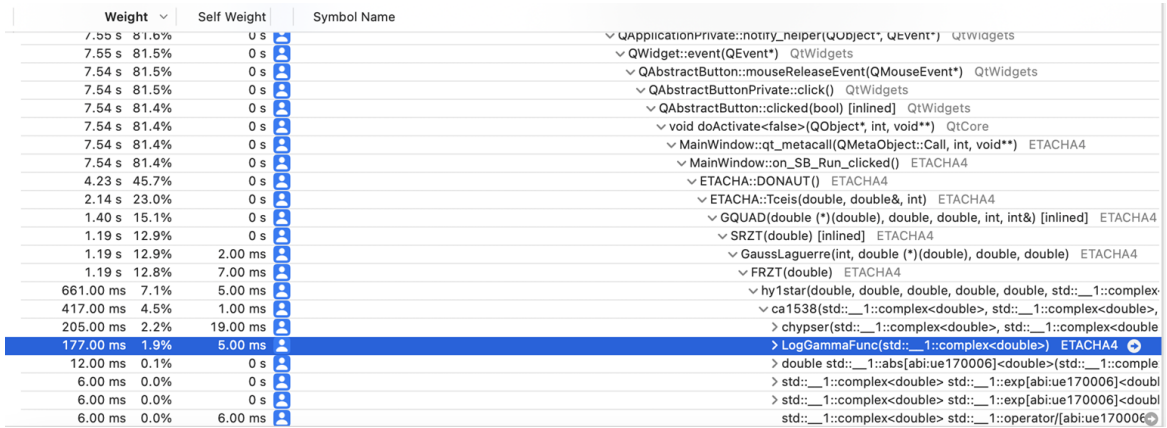
Result #2



Weight	Self Weight	Symbol Name
2.12 s 70.3%	0 s	QAbstractButtonPrivate::click() QtWidgets
2.12 s 70.3%	0 s	QAbstractButton::clicked(bool) [inlined] QtWidgets
2.12 s 70.3%	0 s	void doActivate<false>(QObject*, int, void**) QtCore
2.12 s 70.3%	0 s	MainWindow::qt_metacall(QMetaObject::Call, int, void**) ETACHA4
2.12 s 70.3%	0 s	MainWindow::on_SB_Run_clicked() ETACHA4
2.11 s 70.2%	0 s	ETACHA::DONAUT() ETACHA4
1.08 s 35.8%	0 s	ETACHA::Tceis(double, double&, int) ETACHA4
716.00 ms 23.8%	0 s	GQUAD(double (*)(double), double, double, int, int&) [inlined] ETACHA4
630.00 ms 20.9%	0 s	SRZT(double) [inlined] ETACHA4
630.00 ms 20.9%	0 s	GaussLaguerre(int, double (*)(double), double, double) ETACHA4
628.00 ms 20.8%	1.00 ms	FRZT(double) ETACHA4
353.00 ms 11.7%	1.00 ms	hy1star(double, double, double, double, double, std::__1::complex
222.00 ms 7.3%	0 s	ca1538(std::__1::complex<double>, std::__1::complex<double>,
103.00 ms 3.4%	12.00 ms	chypser(std::__1::complex<double>, std::__1::complex<double
101.00 ms 3.3%	2.00 ms	LogGammaFunc(std::__1::complex<double>) ETACHA4
6.00 ms 0.1%	0 s	double std::__1::abs[abi:ue170006]<double>(std::__1::comple
4.00 ms 0.1%	4.00 ms	std::__1::complex<double> std::__1::operator/[abi:ue170006]<
3.00 ms 0.0%	0 s	std::__1::complex<double> std::__1::exp[abi:ue170006]<doubl
2.00 ms 0.0%	0 s	std::__1::complex<double> std::__1::exp[abi:ue170006]<doubl

-Made three key improvements to make the function run faster and more reliably:

- Use of constexpr for Constants
- Avoided unnecessary pre computations in loop
- Use fewer temporary variables in return statement



Weight	Self Weight	Symbol Name
7.55 s 81.5%	0 s	QWidget::event(QEvent*) QtWidgets
7.54 s 81.5%	0 s	QAbstractButton::mouseReleaseEvent(QMouseEvent*) QtWidgets
7.54 s 81.5%	0 s	QAbstractButtonPrivate::click() QtWidgets
7.54 s 81.4%	0 s	QAbstractButton::clicked(bool) [inlined] QtWidgets
7.54 s 81.4%	0 s	void doActivate<false>(QObject*, int, void**) QtCore
7.54 s 81.4%	0 s	MainWindow::qt_metacall(QMetaObject::Call, int, void**) ETACHA4
7.54 s 81.4%	0 s	MainWindow::on_SB_Run_clicked() ETACHA4
4.23 s 45.7%	0 s	ETACHA::DONAUT() ETACHA4
2.14 s 23.0%	0 s	ETACHA::Tceis(double, double&, int) ETACHA4
1.40 s 15.1%	0 s	GQUAD(double (*)(double), double, double, int, int&) [inlined] ETACHA4
1.19 s 12.9%	0 s	SRZT(double) [inlined] ETACHA4
1.19 s 12.9%	2.00 ms	GaussLaguerre(int, double (*)(double), double, double) ETACHA4
1.19 s 12.8%	7.00 ms	FRZT(double) ETACHA4
661.00 ms 7.1%	5.00 ms	hy1star(double, double, double, double, double, std::__1::complex
417.00 ms 4.5%	1.00 ms	ca1538(std::__1::complex<double>, std::__1::complex<double>,
205.00 ms 2.2%	19.00 ms	chypser(std::__1::complex<double>, std::__1::complex<double
177.00 ms 1.9%	5.00 ms	LogGammaFunc(std::__1::complex<double>) ETACHA4
12.00 ms 0.1%	0 s	double std::__1::abs[abi:ue170006]<double>(std::__1::comple
6.00 ms 0.0%	0 s	std::__1::complex<double> std::__1::exp[abi:ue170006]<doubl
6.00 ms 0.0%	0 s	std::__1::complex<double> std::__1::exp[abi:ue170006]<doubl
6.00 ms 0.0%	6.00 ms	std::__1::complex<double> std::__1::operator/[abi:ue170006

-Results of optimisation

Overall function runtime cut by more than 50%

#3

List of B-N	N~ of Bottleneck w/ line number
e_Tceis	5 from lines 850-859
e_Tceis	Function cGamLn lines 1242-1269
e_Tceis	Function diser lines 1027-1033
e_ETACHA	Function CalcOlegSum lines 1317-1331
e_F4	Function f_numPP line 673
	Function f_numP line 636
	Function f_KK line line 534
e_Tceis	Function digam all lines 879-884

```
nz=xz;
 if(xz != 0.)  divx=nz/xz;
 if(xz == 0. || divx == 1.)
                {
                if(yz == 0.)
                        {
                        cdiser=-gama;
                        for(int k=1; k<=nz-1; k++)
                                cdiser = cdiser+1./k;
                        }
                 else    {
                        digi1(nz,xz,yz,cdiser);
                        }
                }
  else digi2(z,cdiser);
return cdiser;
```

METHODS THAT WE WILL USE TO OPTIMIZE-

-Asymptotic Expansion for Large nz

-Declare xz, yz, and nz as const

-Simplified Conditional Structure

#4

List of B-N	N~ of Bottleneck w/ line number
e_Tceis	5 from lines 850-859
e_Tceis	Function cGamLn lines 1242-1269
e_Tceis	Function diser lines 1027-1033
e_ETACHA	Function CalcOlegSum lines 1317-1331
e_F4	Function f_numPP line 673
	Function f_numP line 636
	Function f_KK line line 534
e_Tceis	Function digam all lines 879-884

```
void CalcOlegSum(double *V)
{
static int counter=0;

    for(int i=0; i<8; i++) sumM[i]=0;

    for(int I=1;   I<=63;   I++) sumM[0] += V[I];
    for(int I=64;  I<=66;   I++) sumM[1] += V[I];
    for(int I=67;  I<=73;   I++) sumM[2] += V[I];
    for(int I=74;  I<=84;   I++) sumM[3] += V[I];
    for(int I=85;  I<=293;  I++) sumM[4] += V[I];
    for(int I=294; I<=326;  I++) sumM[5] += V[I];
    for(int I=327; I<=1283; I++) sumM[6] += V[I];

    for(int i=0; i<7; i++)
                    if(i!=4)
                        sumM[7] += (sumM[i]-1.);
```

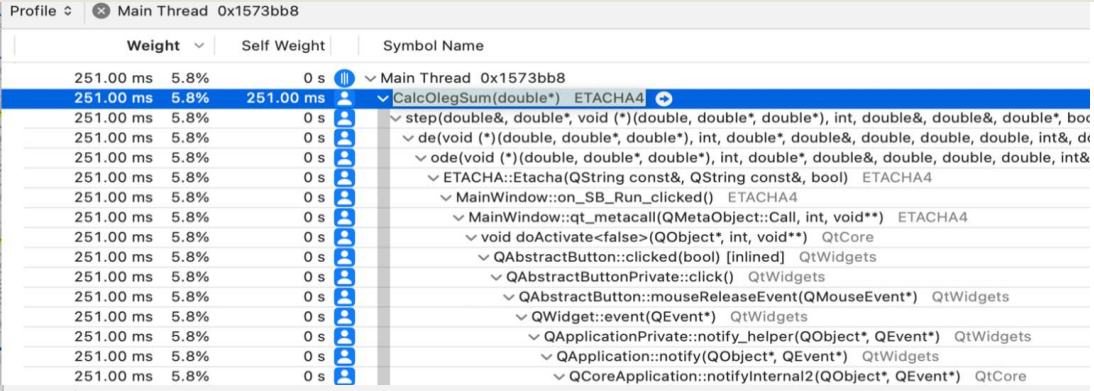
METHODS THAT WE WILL USE TO OPTIMIZE-

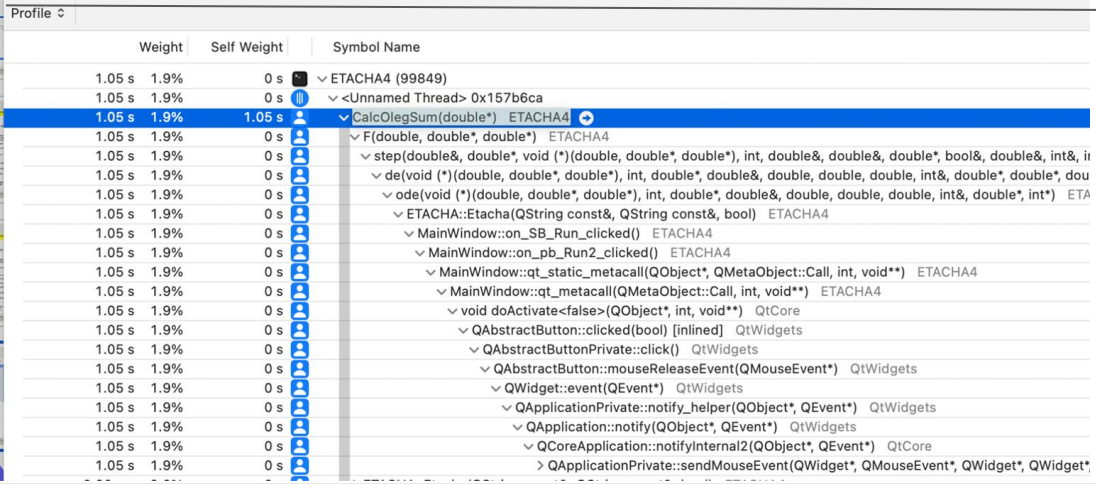
-Reduced Loop Overhead: Instead of looping separately for each sum range, a single loop iterates through V, categorizing values in one pass.

-The categorization is handled via if-else blocks

-Using constexpr for range values

Results #4





Made three key improvements to make the function run faster and more reliably:

- Instead of looping separately for each sum range, a single loop iterates through V, categorizing values in one pass.
- The categorization is handled via if-else blocks

-Results of optimisation

Overall function runtime cut by more than 65%

#5 and #6 and Results

List of B-N	N~ of Bottleneck w/ line number
e_Tceis	5 from lines 850-859
e_Tceis	Function cGamLn lines 1242-1269
e_Tceis	Function diser lines 1027-1033
e_ETACHA	Function CalcOlegSum lines 1317-1331
e_F4	Function f_numPP line 673
	Function f_numP line 636
	Function f_KK line line 534
e_Tceis	Function digam all lines 879-884

```
int f_numPP(int NCO)
{
int numPP=NUM3[NCO+1];
return numPP;
}
```

```
int f_numP(int NCO)
{
int numP=NUM2[NCO+1];
return numP;
}
```

-Results of optimisation

- Removed the unnecessary numPP and numP variable, returned directly
- (f_numPP)-The function runtime has decreased by approximately 29%
- (f_numP)-Function runtime has negligible runtime compared to program runtime(Almost 95% decrease).

#7 and #8 and Results

List of B-N	N~ of Bottleneck w/ line number
e_Tceis	5 from lines 850-859
e_Tceis	Function cGamLn lines 1242-1269
e_Tceis	Function diser lines 1027-1033
e_ETACHA	Function CalcOlegSum lines 1317-1331
e_F4	Function f_numPP line 673
	Function f_numP line 636
	Function f_IKM line 706
	Function f_IN line 700

```
int f_IKM(int N, int num)
{
int f_IKM=IC03[num]-N*100;
return f_IKM;
}
```

```
int f_IN(int num)
{
int f_IN=IC03[num]/100;
return f_IN;
}
```

-Results of optimisation

- Removed the unnecessary f_IKM and f_IN variable, returned directly
- % operator to calculate the remainder instead of subtracting multiples.
- (f_IKM)-The function runtime has decreased by approximately 75%
- (f_IN)-The function runtime has decreased by approximately 75%

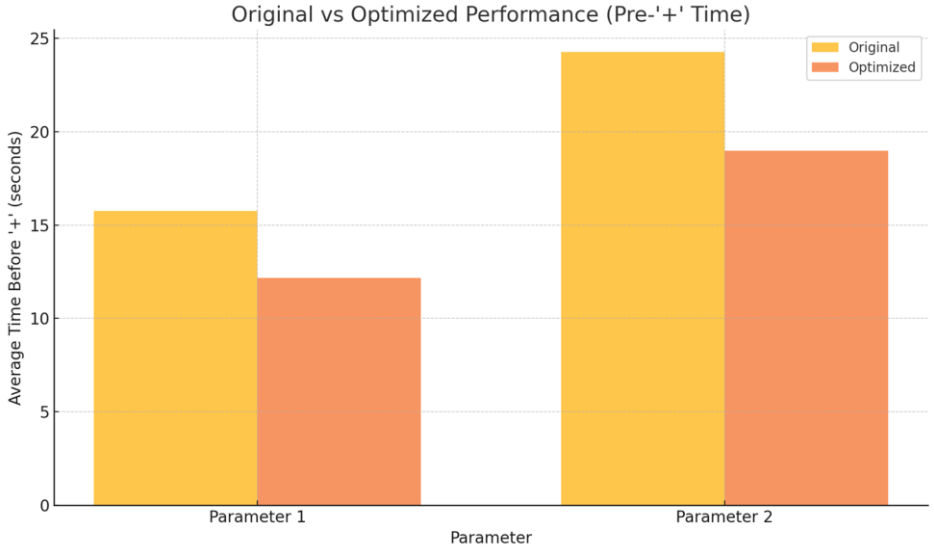
Final Results

Runs

- Each version (optimized & original) was run 3 times with identical parameters(Initial energy, projectile, target etc.) to ensure fairness.

- I averaged the run times and found the time profit between the two versions.

Parameter	Optimized	Original	Time Saved
Parameter 1	13.601666666666700	17.171333333333300	3.5696666666666700
Parameter 2	41.03966666666670	46.76866666666670	5.729000000000000
Parameter 3	22.16033333333330	22.664666666666700	0.5043333333333350
Parameter 4	1.458	1.492	0.03400000000000000



Results

- Parameters 3 and 4 were run on Ionization and Excitation models being set on PWBA(fast). **We see only around a 2.2% time profit.**

- Parameters 1 and 2 were not run on Ionization and Excitation models set on PWBA(fast). Versions v.3 and v.4 were used and both were tested with both integration models. **We saw a huge time profit in this case(around 23%).** The profit was mainly due to faster cross-section calculation.

- The time profit percentages were around the same, suggesting that **the optimized code works for all cases.**