Software Optimization Case Study

Yu-Ping Zhao
Yuping.zhao@intel.com

Agenda

RELION Background

RELION ITAC and VTUE Analyze

RELION Auto-Refine Workload Optimization

RELION 2D Classification Workload Optimization

Further Optimization

Background

- Cryo-electron microscopy (cryo-EM), is a form of transmission electron microscopy (TEM) where the sample is studied at cryogenic temperatures
- RELION (for **RE**gularised **LI**kelihood **O**ptimisatio**N**, pronounce *rely-on*) is a stand-alone computer program that employs an empirical Bayesian approach to refinement of (multiple) 3D reconstructions or 2D class averages in electron cryo-microscopy (cryo-EM).
- http://www2.mrc-lmb.cam.ac.uk/relion/index.php/Main_Page
- Relion use MPI (Message Passing Interface) for distributed-memory parallelisation, and POSIX threads for shared-memory parallelisation

Agenda

RELION Background

RELION ITAC and VTUE Analyze

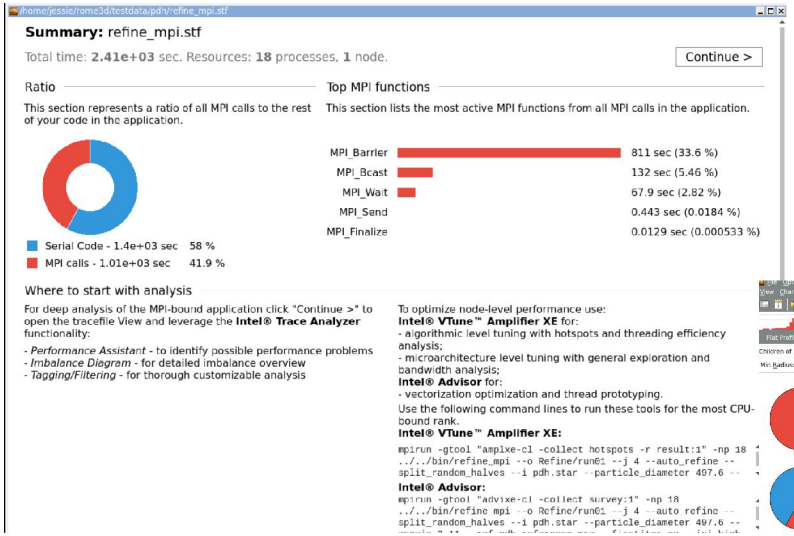
RELION Auto-Refine Workload Optimization

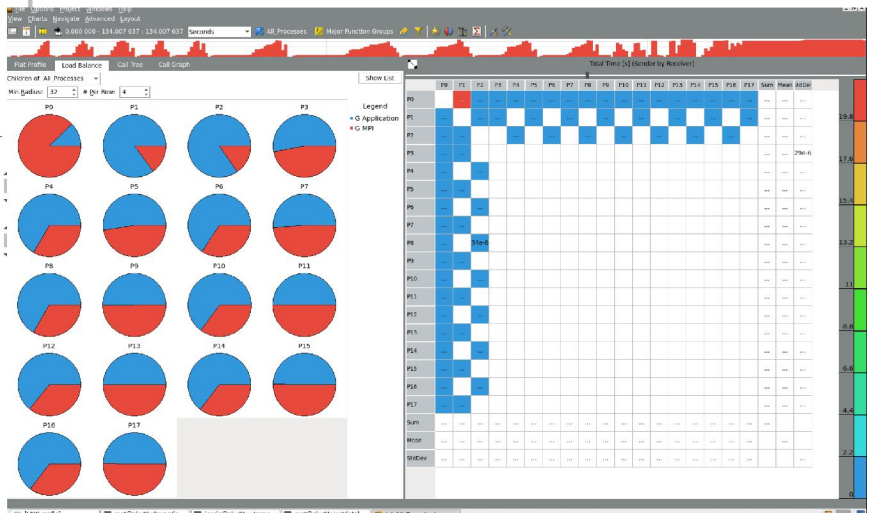
RELION 2D Classification Workload Optimization

Further Optimization

RELION ITAC Report

Load balance of MPI process is not very good

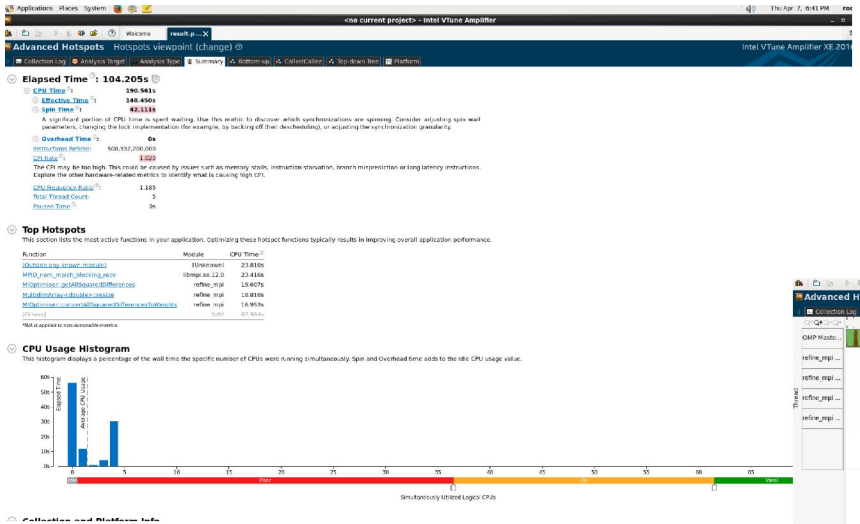


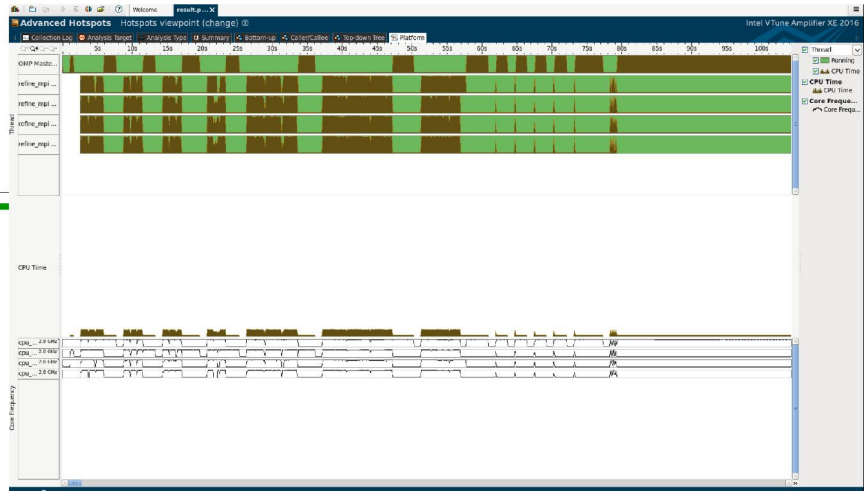


RELION VTune Analyze

High Spin time(mostly focused on MPI Communication)

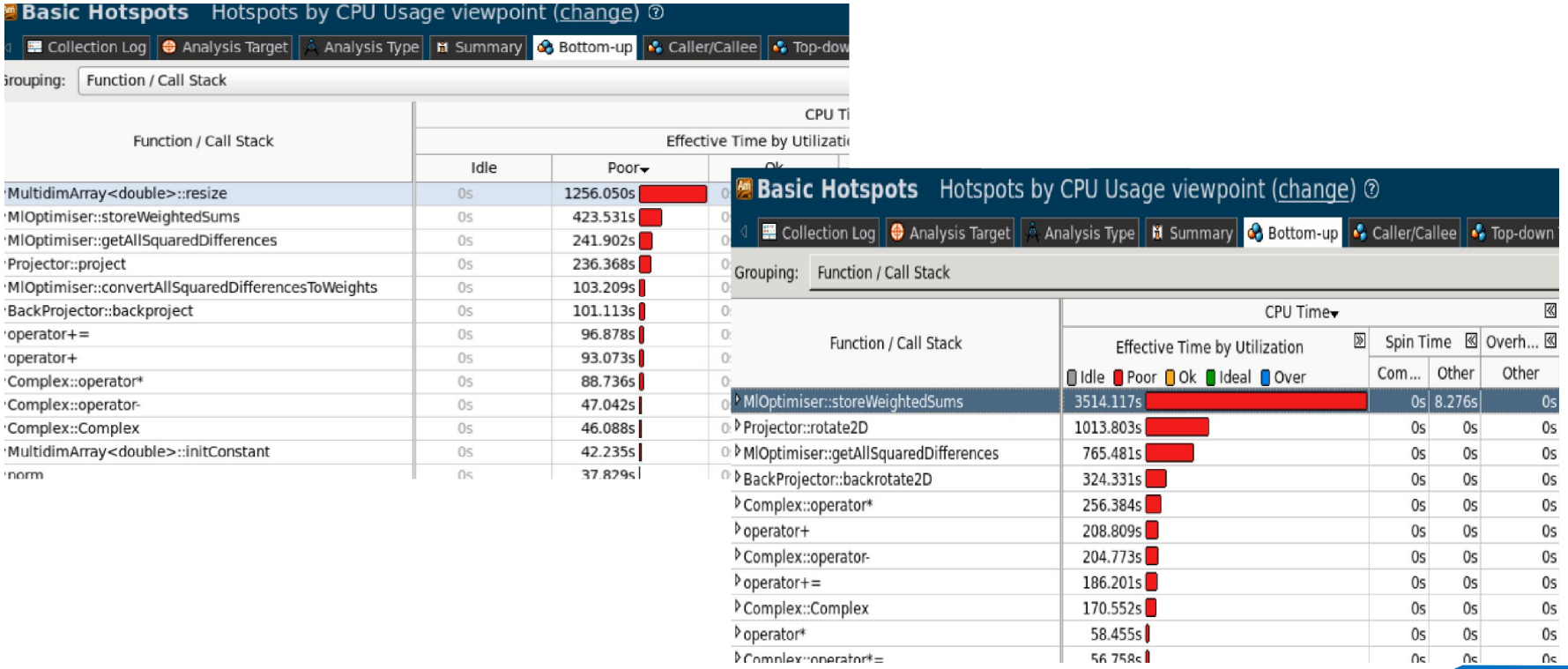
CPU usage is not balanced between processes





RELION different workload VTune Analyze

3D Auto Refine hotspots and 2D Classification hotspots



Agenda

RELION Background

RELION VTUE Analyze

RELION Auto-Refine Workload Optimization

RELION 2D Classification Workload Optimization

Further Optimization

Optimization(1) –Data Alignment

Align memory allocation with 64 byte

Reload new() and delete() in multidim_array.h, it helps to other optimization

```
501     void *operator new(size_t size)
502     {
503         return _mm_malloc(size, 64);
504     }
505     void operator delete(void *p)
506     {
507         _mm_free(p);
508     }
```

Optimization(2) – Vector No.1 hotspot

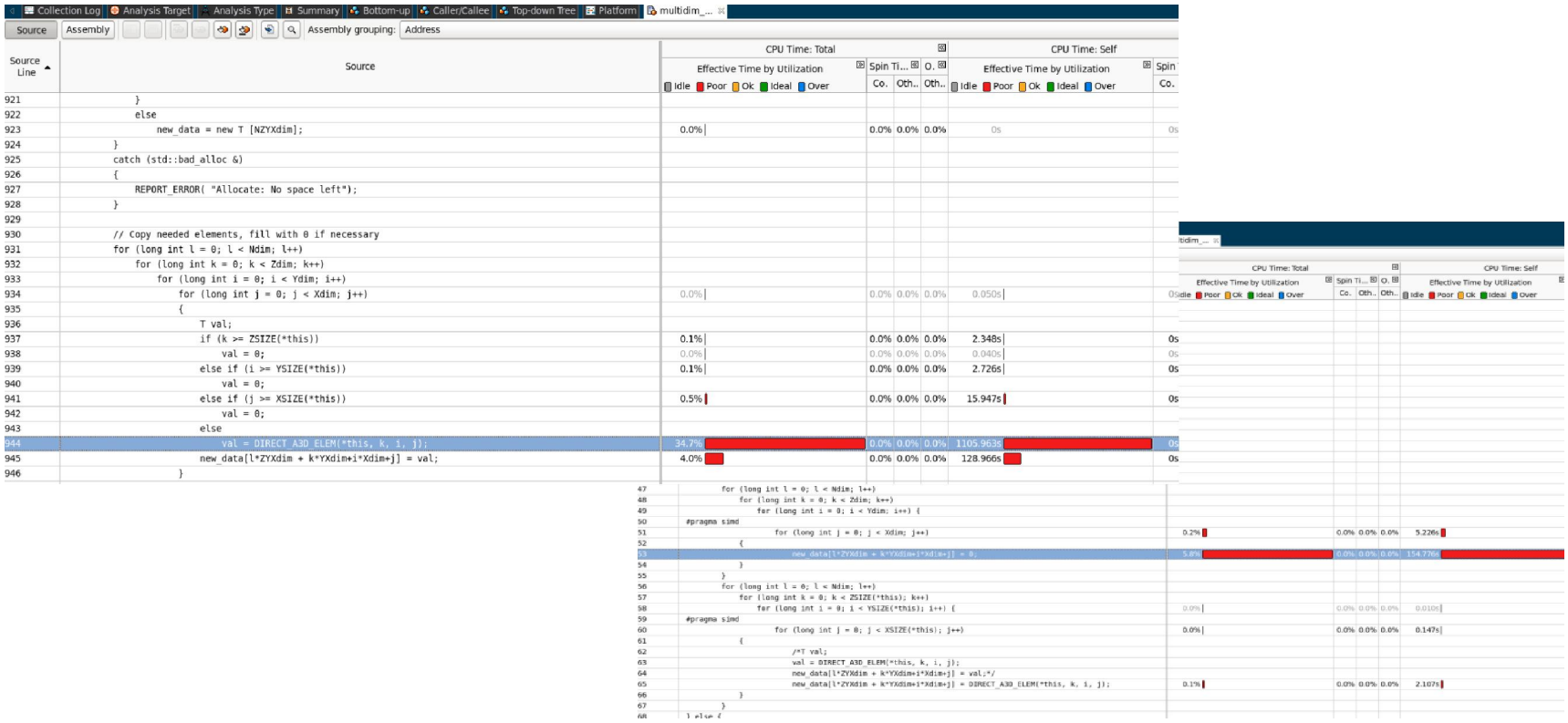
Optimized code(Major loop was vectored with SIMD successfully)

```
946 if ( (ZSIZE(*this)<= Zdim) && (YSIZE(*this)<= Ydim) && (XSIZE(*this)<= Xdim) ) {
947         for (long int l = 0; l < Ndim; l++)
948             for (long int k = 0; k < Zdim; k++)
949                 for (long int i = 0; i < Ydim; i++) {
950 #pragma simd
951                     for (long int j = 0; j < Xdim; j++)
952                     {
953                         new_data[l*ZYXdim + k*YXdim+i*Xdim+j] = 0;
954                     }
955                 }
956         for (long int l = 0; l < Ndim; l++)
957             for (long int k = 0; k < ZSIZE(*this); k++)
958                 for (long int i = 0; i < YSIZE(*this); i++) {
959 #pragma simd
960                     for (long int j = 0; j < XSIZE(*this); j++)
961                     {
962                         /*T val;
963                         val = DIRECT_A3D_ELEM(*this, k, i, j);
964                         new_data[l*ZYXdim + k*YXdim+i*Xdim+j] = val;*/
965                         new_data[l*ZYXdim + k*YXdim+i*Xdim+j] = DIRECT_A3D_ELEM(*this, k, i, j);
966                     }
967                 }
968 } else {
969
970         for (long int l = 0; l < Ndim; l++)
971             for (long int k = 0; k < Zdim; k++)
972                 for (long int i = 0; i < Ydim; i++)
973                     for (long int j = 0; j < Xdim; j++)
974                     {
975                         T val;
976                         if (k >= ZSIZE(*this))
977                             val = 0;
978                         else if (i >= YSIZE(*this))
979                             val = 0;
980                         else if (j >= XSIZE(*this))
981                             val = 0;
982                         else
983                             val = DIRECT_A3D_ELEM(*this, k, i, j);
984                         new_data[l*ZYXdim + k*YXdim+i*Xdim+j] = val;
985                     }
986 }
```

Original code

```
931         for (long int l = 0; l < Ndim; l++)
932             for (long int k = 0; k < Zdim; k++)
933                 for (long int i = 0; i < Ydim; i++)
934                     for (long int j = 0; j < Xdim; j++)
935                     {
936                         T val;
937                         if (k >= ZSIZE(*this))
938                             val = 0;
939                         else if (i >= YSIZE(*this))
940                             val = 0;
941                         else if (j >= XSIZE(*this))
942                             val = 0;
943                         else
944                             val = DIRECT_A3D_ELEM(*this, k, i, j);
945                         new_data[l*ZYXdim + k*YXdim+i*Xdim+j] = val;
946                     }
```

Intel Vtune Analyze Compare(Before/After Optimize)



RELION Optimization(3) –inline function

Inline function to make it be vectorized,

```
Complex *opp;
FOR_ALL_DIRECT_ELEMENTS_IN_MULTIDIMARRAY(Frefctf)
{
        diff2 -= (DIRECT_MULTIDIM_ELEM(Frefctf, n)).real * (*(Fimg_shift + n)).real;
        diff2 -= (DIRECT_MULTIDIM_ELEM(Frefctf, n)).imag * (*(Fimg_shift + n)).imag;
        opp = & DIRECT_MULTIDIM_ELEM(Frefctf, n);
        suma2 += opp->real * opp->real + opp->imag * opp->imag;
        //suma2 += norm(DIRECT_MULTIDIM_ELEM(Frefctf, n));
}
```

#vi src/ml_optimizer.optrpt

Before Vectorize

LOOP BEGIN at src/ml_optimiser.cpp(3652,13)

remark #15523: loop was not vectorized: loop control variable n was found, but loop iteration count cannot be computed before executing the loop

LOOP END

After Vectorize

LOOP BEGIN at src/ml_optimiser.cpp(3654,97)

... ...

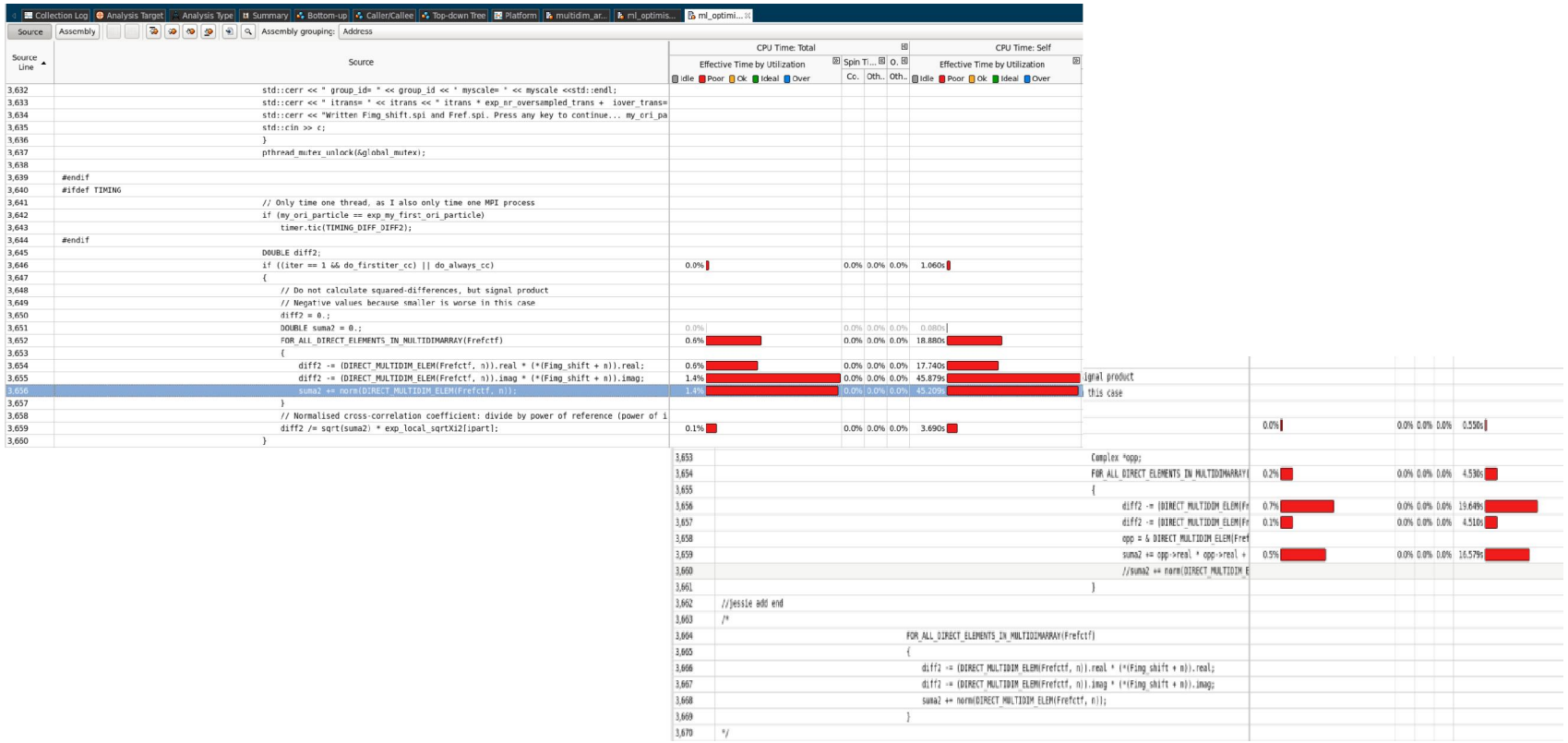
remark #15300: LOOP WAS VECTORIZED

.. ...

remark #15478: estimated potential speedup: 3.990

LOOP END

Intel Vtune Hotspots Analyze



Agenda

RELION Background

RELION ITAC and VTUE Analyze

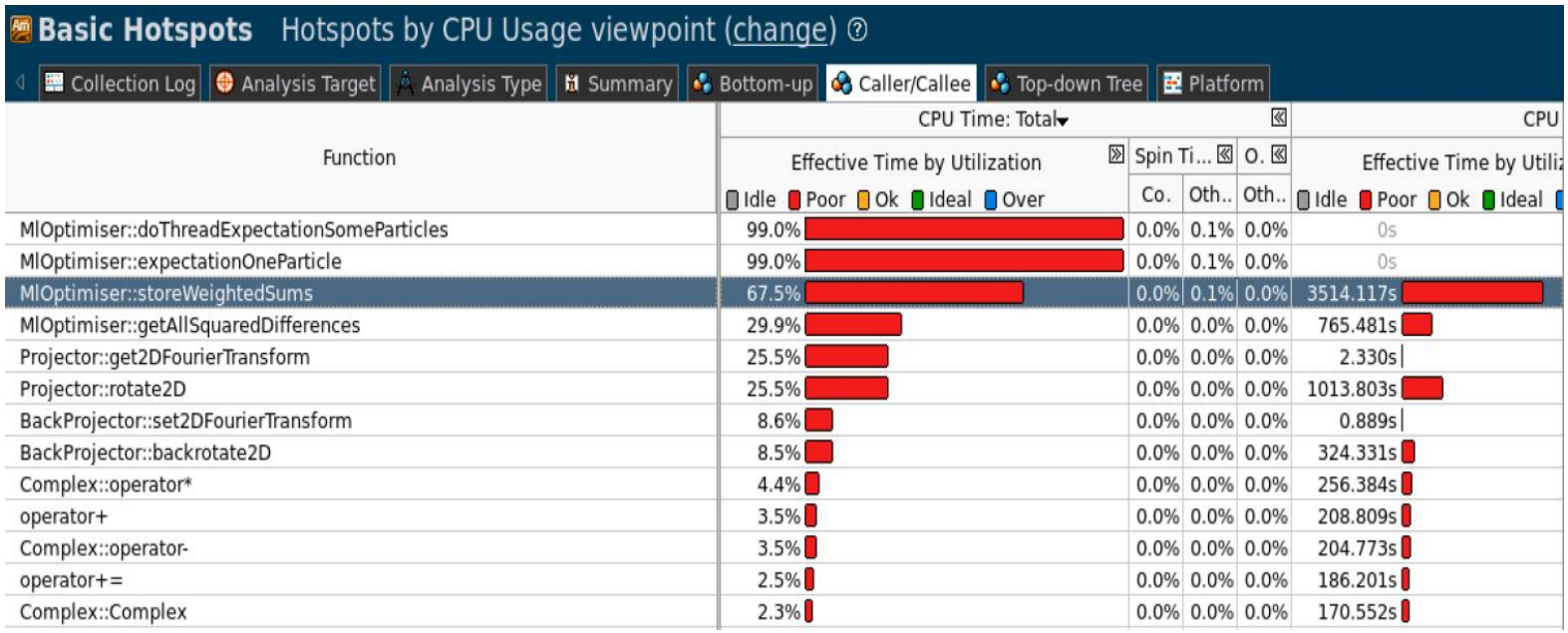
RELION Auto-Refine Workload Optimization

RELION 2D Classification Workload Optimization

Further Optimization

Optimize(1) –Remove vector dependence

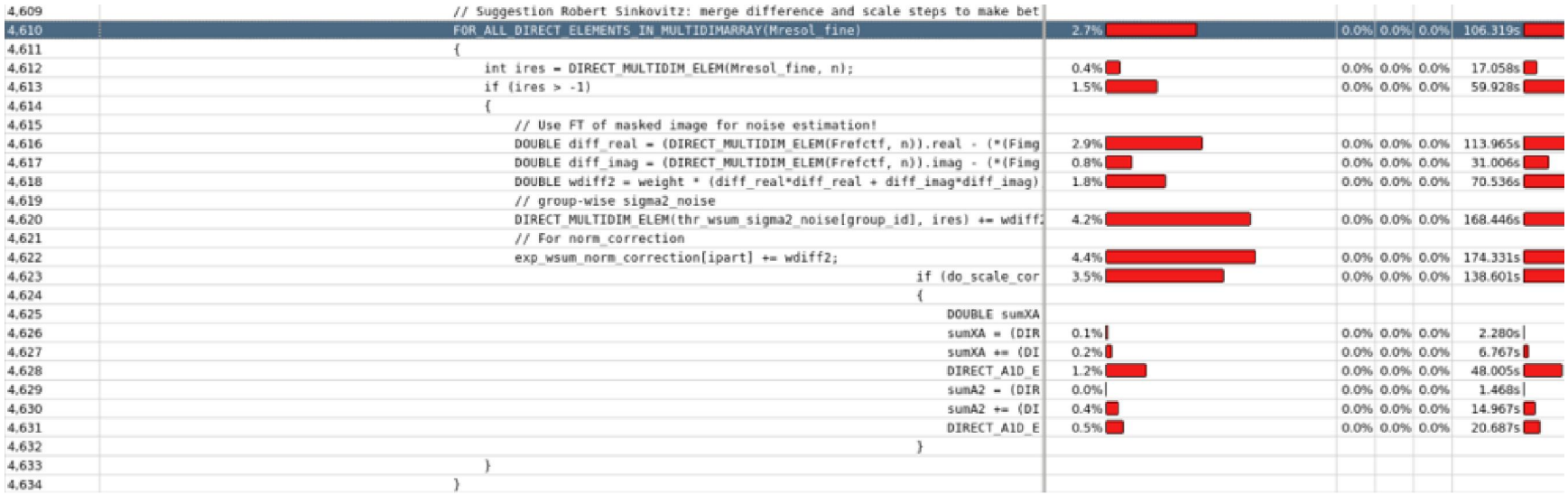
Get hotspots info from Vtune analyze



Function	CPU Time: Total					CPU
	Effective Time by Utilization		Spin Ti...	O.		Effective Time by Utili
	Idle Poor Ok Ideal Over		Co.	Oth..	Oth..	Idle Poor Ok Ideal
MlOptimiser::doThreadExpectationSomeParticles	99.0%		0.0%	0.1%	0.0%	0s
MlOptimiser::expectationOneParticle	99.0%		0.0%	0.1%	0.0%	0s
MlOptimiser::storeWeightedSums	67.5%		0.0%	0.1%	0.0%	3514.117s
MlOptimiser::getAllSquaredDifferences	29.9%		0.0%	0.0%	0.0%	765.481s
Projector::get2DFourierTransform	25.5%		0.0%	0.0%	0.0%	2.330s
Projector::rotate2D	25.5%		0.0%	0.0%	0.0%	1013.803s
BackProjector::set2DFourierTransform	8.6%		0.0%	0.0%	0.0%	0.889s
BackProjector::backrotate2D	8.5%		0.0%	0.0%	0.0%	324.331s
Complex::operator*	4.4%		0.0%	0.0%	0.0%	256.384s
operator+	3.5%		0.0%	0.0%	0.0%	208.809s
Complex::operator-	3.5%		0.0%	0.0%	0.0%	204.773s
operator+=	2.5%		0.0%	0.0%	0.0%	186.201s
Complex::Complex	2.3%		0.0%	0.0%	0.0%	170.552s

Optimize(1) - Remove vector dependence

Line 4610: for loop is not vectorized



```
4,609                   // Suggestion Robert Sinkovitz: merge difference and scale steps to make bet
4,610                   FOR_ALL_DIRECT_ELEMENTS_IN_MULTIDIMARRAY(Mresol_fine)                    2.7%   0.0% 0.0% 0.0% 106.319s
4,611                   {
4,612                       int ires = DIRECT_MULTIDIM_ELEM(Mresol_fine, n);                     0.4%   0.0% 0.0% 0.0% 17.058s
4,613                       if (ires > -1)                                                       1.5%   0.0% 0.0% 0.0% 59.928s
4,614                       {
4,615                           // Use FT of masked image for noise estimation!
4,616                           DOUBLE diff_real = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).real - (*(Fimg 2.9%   0.0% 0.0% 0.0% 113.965s
4,617                           DOUBLE diff_imag = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).imag - (*(Fimg 0.8%   0.0% 0.0% 0.0% 31.006s
4,618                           DOUBLE wdiff2 = weight * (diff_real*diff_real + diff_imag*diff_imag) 1.8%   0.0% 0.0% 0.0% 70.536s
4,619                           // group-wise sigma2_noise
4,620                           DIRECT_MULTIDIM_ELEM(thr_wsum_sigma2_noise[group_id], ires) += wdiff2 4.2%   0.0% 0.0% 0.0% 168.446s
4,621                           // For norm_correction
4,622                           exp_wsum_norm_correction[ipart] += wdiff2;                       4.4%   0.0% 0.0% 0.0% 174.331s
4,623                                                                   if (do_scale_cor         3.5%   0.0% 0.0% 0.0% 138.601s
4,624                                                                   {
4,625                                                                       DOUBLE sumXA
4,626                                                                       sumXA = (DIR         0.1%   0.0% 0.0% 0.0% 2.280s
4,627                                                                       sumXA += (DI         0.2%   0.0% 0.0% 0.0% 6.767s
4,628                                                                       DIRECT_A1D_E         1.2%   0.0% 0.0% 0.0% 48.005s
4,629                                                                       sumA2 = (DIR         0.0%   0.0% 0.0% 0.0% 1.468s
4,630                                                                       sumA2 += (DI         0.4%   0.0% 0.0% 0.0% 14.967s
4,631                                                                       DIRECT_A1D_E         0.5%   0.0% 0.0% 0.0% 20.687s
4,632                                                                   }
4,633                       }
4,634                   }
```

```
LOOP BEGIN at src/ml_optimiser.cpp(4610,12)
   remark #15344: loop was not vectorized: vector dependence prevents vectorization
   remark #15346: vector dependence: assumed ANTI dependence between this line 4612 and exp_wsum_scale_correction_AA.__b line 4631
   remark #15346: vector dependence: assumed FLOW dependence between exp_wsum_scale_correction_AA.__b line 4631 and this line 4612
LOOP END
```

Optimize(1) - Remove vector dependence

Split hot loop into three parts, vectorized the first two loops.

```
4,641  #pragma simd reduction(+:inner_exp_wsum_norm_correction)
4,642      for (long int Mresol_initer=0; Mresol_initer<Mresol_inner_size; Mresol_initer++) {
4,643          long int n = Mresol_outiter*Mresol_inner_size + Mresol_initer;
4,644          int ires = DIRECT_MULTIDIM_ELEM(Mresol_fine, n);
4,645          if (ires > -1)
4,646          {
4,647              // Use FT of masked image for noise estimation!
4,648              DOUBLE diff_real = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).real - (*(Fimg_shift + n)).rea
4,649              DOUBLE diff_imag = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).imag - (*(Fimg_shift + n)).ima
4,650              Mresol_fine_wdiff2[Mresol_initer] = weight * (diff_real*diff_real + diff_imag*diff_i
4,651              // For norm_correction
4,652              //exp_wsum_norm_correction[ipart] += Mresol_fine_wdiff2[Mresol_initer];
4,653              inner_exp_wsum_norm_correction += Mresol_fine_wdiff2[Mresol_initer];
4,654          }
4,655      }
4,656  #pragma ivdep
4,657      for (long int Mresol_initer=0; Mresol_initer<Mresol_inner_size; Mresol_initer++) {
4,658          long int n = Mresol_outiter*Mresol_inner_size + Mresol_initer;
4,659          int ires = DIRECT_MULTIDIM_ELEM(Mresol_fine, n);
4,660          //int ires1 = DIRECT_MULTIDIM_ELEM(Mresol_fine, n+1);
4,661          //_mm_prefetch((char *)(&DIRECT_A1D_ELEM(mymodel.data_vs_prior_class[exp_iclass], ires1)
4,662          if (ires > -1)
4,663          {
4,664              // group-wise sigma2_noise
4,665              //DIRECT_MULTIDIM_ELEM(thr_wsum_sigma2_noise[group_id], ires) += Mresol_fine_wdiff2[
4,666              Mresol_fine_sigma2_noise[n] += Mresol_fine_wdiff2[Mresol_initer];
4,667          }
4,668      }
4,669
4,670  }//end for Mresol_outier first loop
```

4,642	1.4%	0.0%	0.0%	0.0.	40.074s
4,644	0.0%	0.0%	0.0%	0.0.	0.350s
4,645	0.7%	0.0%	0.0%	0.0.	20.269s
4,648	7.1%	0.0%	0.0%	0.0.	204.495s
4,649	0.2%	0.0%	0.0%	0.0.	4.621s
4,650	2.2%	0.0%	0.0%	0.0.	63.237s
4,653	0.3%	0.0%	0.0%	0.0.	8.524s
4,662	0.3%	0.0%	0.0%	0.0.	7.929s
4,666	1.1%	0.0%	0.0%	0.0.	32.277s

Optimize(1) - Remove vector dependence

Reduce the loop number of third loop

```
4,697       for(Mresol_fine_data_vs_prior_class_idx=0; Mresol_fine_data_vs_prior_class_idx<Mresol_fi   1.6%   0.0% 0.0% 0.0.   44.698s
4,698           int n = Mresol_fine_data_vs_prior_class[Mresol_fine_data_vs_prior_class_idx];           0.4%   0.0% 0.0% 0.0.   10.909s
4,699           int ires = Mresol_fine_data_vs_prior_class_ires[Mresol_fine_data_vs_prior_class_idx]    0.7%   0.0% 0.0% 0.0.   19.989s
4,700               DOUBLE sumXA, sumA2;
4,701               sumXA = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).real * (*(Fimg_shift + n)).real;        1.2%   0.0% 0.0% 0.0.   35.711s
4,702               sumXA += (DIRECT_MULTIDIM_ELEM(Frefctf, n)).imag * (*(Fimg_shift + n)).imag;       0.3%   0.0% 0.0% 0.0.   7.493s
4,703               DIRECT_A1D_ELEM(exp_wsum_scale_correction_XA[ipart], ires) += weight * sumXA;      3.8%   0.0% 0.0% 0.0.   108.124s
4,704               sumA2 = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).real * (DIRECT_MULTIDIM_ELEM(Frefctf,  0.1%   0.0% 0.0% 0.0.   2.181s
4,705               sumA2 += (DIRECT_MULTIDIM_ELEM(Frefctf, n)).imag * (DIRECT_MULTIDIM_ELEM(Frefctf  1.0%   0.0% 0.0% 0.0.   30.138s
4,706               DIRECT_A1D_ELEM(exp_wsum_scale_correction_AA[ipart], ires) += weight * sumA2;      0.9%   0.0% 0.0% 0.0.   26.390s
4,707       }
```

Optimize (1)-Remove vector dependence

Vector report, First and second loops was vectorized

```
LOOP BEGIN at src/ml_optimiser.cpp(4640,9)
   remark #15389: vectorization support: reference Frefctf.data has unaligned access   [ src/ml_optimiser.cpp(4645,86) ]
   remark #15389: vectorization support: reference Fimg_shift has unaligned access   [ src/ml_optimiser.cpp(4645,86) ]
   remark #15388: vectorization support: reference Mresol_fine_wdiff2 has aligned access   [ src/ml_optimiser.cpp(4647,22) ]
   remark #15388: vectorization support: reference Mresol_fine_wdiff2 has aligned access   [ src/ml_optimiser.cpp(4650,21) ]
   remark #15381: vectorization support: unaligned access used inside loop body
   remark #15305: vectorization support: vector length 8
   remark #15309: vectorization support: normalized vectorization overhead 0.661
   remark #15301: SIMD LOOP WAS VECTORIZED
   remark #15450: unmasked unaligned unit stride loads: 1
   remark #15452: unmasked strided loads: 4
   remark #15454: masked aligned unit stride loads: 1
   remark #15455: masked aligned unit stride stores: 1
   remark #15456: masked unaligned unit stride loads: 2
   remark #15475: --- begin vector loop cost summary ---
   remark #15476: scalar loop cost: 29
   remark #15477: vector loop cost: 7.370
   remark #15478: estimated potential speedup: 3.710
   remark #15488: --- end vector loop cost summary ---
LOOP END
```

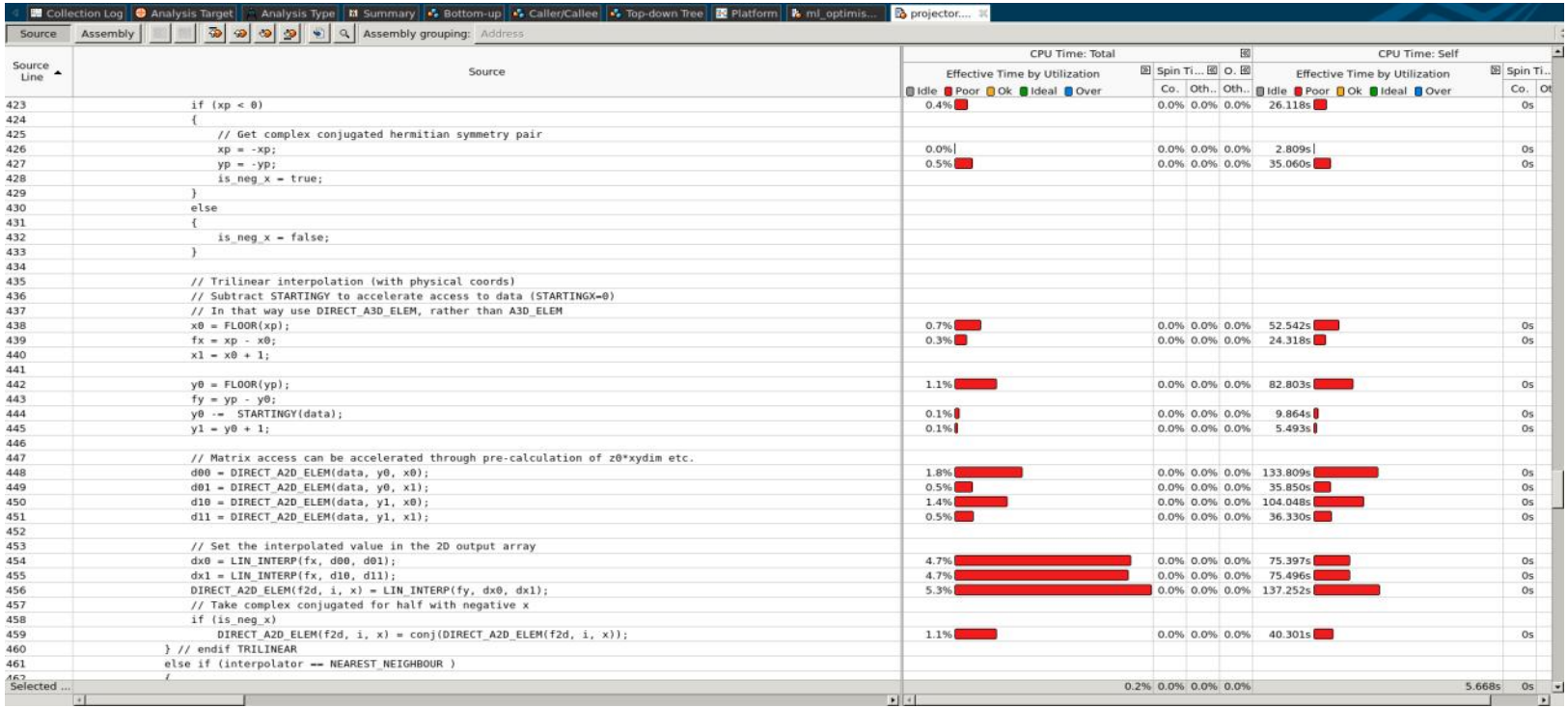
Optimize (2)-Remove exception calls

Get hotspots info from Vtune analyze

MlOptimiser::expectationOneParticle	99.0%	0.0%	0.1%	0.0%	0s
MlOptimiser::storeWeightedSums	67.5%	0.0%	0.1%	0.0%	3514.117s
MlOptimiser::getAllSquaredDifferences	29.9%	0.0%	0.0%	0.0%	765.481s
Projector::get2DFourierTransform	25.5%	0.0%	0.0%	0.0%	2.330s
Projector::rotate2D	25.5%	0.0%	0.0%	0.0%	1013.803s
BackProjector::set2DFourierTransform	8.6%	0.0%	0.0%	0.0%	0.889s
BackProjector::backrotate2D	8.5%	0.0%	0.0%	0.0%	324.331s
Complex::operator*	4.4%	0.0%	0.0%	0.0%	256.384s
operator+	3.5%	0.0%	0.0%	0.0%	208.809s
Complex::operator-	3.5%	0.0%	0.0%	0.0%	204.773s
operator+=	2.5%	0.0%	0.0%	0.0%	186.201s
Complex::Complex	2.3%	0.0%	0.0%	0.0%	170.552s

Optimize (2)-Remove exception calls

Line 410: For loop is not vectorized



Source Line	Source	CPU Time: Total — Effective Time by Utilization (Idle Poor Ok Ideal Over)	CPU Time: Total — Spin Ti...	CPU Time: Total — O.	CPU Time: Total — Oth..	CPU Time: Self — Effective Time by Utilization (Idle Poor Ok Ideal Over)	CPU Time: Self — Spin Ti..
423	if (xp < 0)	0.4%	0.0%	0.0%	0.0%	26.118s	0s
424	{						
425	// Get complex conjugated hermitian symmetry pair						
426	xp = -xp;	0.0%	0.0%	0.0%	0.0%	2.809s	0s
427	yp = -yp;	0.5%	0.0%	0.0%	0.0%	35.060s	0s
428	is_neg_x = true;						
429	}						
430	else						
431	{						
432	is_neg_x = false;						
433	}						
434							
435	// Trilinear interpolation (with physical coords)						
436	// Subtract STARTINGY to accelerate access to data (STARTINGX=0)						
437	// In that way use DIRECT_A3D_ELEM, rather than A3D_ELEM						
438	x0 = FLOOR(xp);	0.7%	0.0%	0.0%	0.0%	52.542s	0s
439	fx = xp - x0;	0.3%	0.0%	0.0%	0.0%	24.318s	0s
440	x1 = x0 + 1;						
441							
442	y0 = FLOOR(yp);	1.1%	0.0%	0.0%	0.0%	82.803s	0s
443	fy = yp - y0;						
444	y0 -= STARTINGY(data);	0.1%	0.0%	0.0%	0.0%	9.864s	0s
445	y1 = y0 + 1;	0.1%	0.0%	0.0%	0.0%	5.493s	0s
446							
447	// Matrix access can be accelerated through pre-calculation of z0*xydim etc.						
448	d00 = DIRECT_A2D_ELEM(data, y0, x0);	1.8%	0.0%	0.0%	0.0%	133.809s	0s
449	d01 = DIRECT_A2D_ELEM(data, y0, x1);	0.5%	0.0%	0.0%	0.0%	35.850s	0s
450	d10 = DIRECT_A2D_ELEM(data, y1, x0);	1.4%	0.0%	0.0%	0.0%	104.048s	0s
451	d11 = DIRECT_A2D_ELEM(data, y1, x1);	0.5%	0.0%	0.0%	0.0%	36.330s	0s
452							
453	// Set the interpolated value in the 2D output array						
454	dx0 = LIN_INTERP(fx, d00, d01);	4.7%	0.0%	0.0%	0.0%	75.397s	0s
455	dx1 = LIN_INTERP(fx, d10, d11);	4.7%	0.0%	0.0%	0.0%	75.496s	0s
456	DIRECT_A2D_ELEM(f2d, i, x) = LIN_INTERP(fy, dx0, dx1);	5.3%	0.0%	0.0%	0.0%	137.252s	0s
457	// Take complex conjugated for half with negative x						
458	if (is_neg_x)						
459	DIRECT_A2D_ELEM(f2d, i, x) = conj(DIRECT_A2D_ELEM(f2d, i, x));	1.1%	0.0%	0.0%	0.0%	40.301s	0s
460	} // endif TRILINEAR						
461	else if (interpolator == NEAREST_NEIGHBOUR)						
462	{						
Selected ...		0.2%	0.0%	0.0%	0.0%	5.668s	0s

Optimize (2)-Remove exception calls

```
410             for (int x=0; x <= my_r_max; x++)
411             {
412             // Only include points with radius < max_r (exclude points outside circle in square)
413                     r2 = x * x + y2;
414                     if (r2 > max_r2)
415                             continue;
416
417                     // Get logical coordinates in the 3D map
418                     xp = Ainv(0,0) * x + Ainv(0,1) * y;
419                     yp = Ainv(1,0) * x + Ainv(1,1) * y;
420                     if (interpolator == TRILINEAR || r2 < min_r2_nn)
421                     {
422                             // Only asymmetric half is stored
423                             if (xp < 0)
424                             {
425                                     // Get complex conjugated hermitian symmetry pair
426                                     xp = -xp;
427                                     yp = -yp;
428                                     is_neg_x = true;
429                             }
430                             else
431                             {
432                                     is_neg_x = false;
433                             }
434
435                             // Trilinear interpolation (with physical coords)
436                             // Subtract STARTINGY to accelerate access to data (STARTINGX=0)
437                             // In that way use DIRECT_A3D_ELEM, rather than A3D_ELEM
438                     x0 = FLOOR(xp);
439                             fx = xp - x0;
440                             x1 = x0 + 1;
441
442                             y0 = FLOOR(yp);
443                             fy = yp - y0;
444                             y0 -=  STARTINGY(data);
445                             y1 = y0 + 1;
446
447                             // Matrix access can be accelerated through pre-calculation of z0*xydim etc.
448                             d00 = DIRECT_A2D_ELEM(data, y0, x0);
449                             d01 = DIRECT_A2D_ELEM(data, y0, x1);
450                             d10 = DIRECT_A2D_ELEM(data, y1, x0);
451                             d11 = DIRECT_A2D_ELEM(data, y1, x1);
452
453                             // Set the interpolated value in the 2D output array
454                             dx0 = LIN_INTERP(fx, d00, d01);
455                             dx1 = LIN_INTERP(fx, d10, d11);
456                             DIRECT_A2D_ELEM(f2d, i, x) = LIN_INTERP(fy, dx0, dx1);
457                             // Take complex conjugated for half with negative x
458                             if (is_neg_x)
459                                     DIRECT_A2D_ELEM(f2d, i, x) = conj(DIRECT_A2D_ELEM(f2d, i, x));
460                     } // endif TRILINEAR
461                     else if (interpolator == NEAREST_NEIGHBOUR )
462                     {
463                             x0 = ROUND(xp);
464                             y0 = ROUND(yp);
465                             if (x0 < 0)
466                                     DIRECT_A2D_ELEM(f2d, i, x) = conj(A2D_ELEM(data, -y0, -x0));
467                             else
468                                     DIRECT_A2D_ELEM(f2d, i, x) = A2D_ELEM(data, y0, x0);
469                     } // endif NEAREST_NEIGHBOUR
470                     else
471                             REPORT_ERROR("Unrecognized interpolator in Projector::project");
472             } // endif x-loop
```

```
LOOP BEGIN at src/projector.cpp(410,3)
   remark #15333: loop was not vectorized: exception handling for a call prevents vectorization   [ src/projector.cpp(466,40) ]
LOOP END
```

Optimize (2)-Remove exception calls

Vectorized the for loop

```
For(int i=0;i<YSIZE(f2d);i++){

      #pragma ivdep
      for (intx=0;x< min_x;x++){ ...}

      if (interpolator != NEAREST_NEIGHBOUR && interpolator != TRILINEAR) {
            REPORT_ERROR("Unrecognized interpolator in Projector::project");              }

      if (interpolator == TRILINEAR) {
        #pragma ivdep
      for (int x=min_x; x < max_x; x++){}
       }
      if (interpolator == NEAREST_NEIGHBOUR ) {
      #pragma ivdep
      for (int x=min_x; x < max_x; x++){}
      }
}
```

Optimize (2)-Remove exception calls

Vector report, First two loops are vectorized

```
LOOP BEGIN at src/projector.cpp(429,3)
   remark #15415: vectorization support: gather was generated for the variable this:  indirect access, 64bit indexed   [ src/projector.cpp(464,16) ]
   remark #15415: vectorization support: gather was generated for the variable this:  indirect access, 64bit indexed   [ src/projector.cpp(464,54) ]
   remark #15415: vectorization support: gather was generated for the variable this:  indirect access, 64bit indexed   [ src/projector.cpp(464,89) ]
   remark #15415: vectorization support: gather was generated for the variable this:  indirect access, 64bit indexed   [ src/projector.cpp(465,16) ]
   remark #15415: vectorization support: gather was generated for the variable this:  indirect access, 64bit indexed   [ src/projector.cpp(465,54) ]
   remark #15415: vectorization support: gather was generated for the variable this:  indirect access, 64bit indexed   [ src/projector.cpp(465,89) ]
   remark #15415: vectorization support: gather was generated for the variable this:  indirect access, 64bit indexed   [ src/projector.cpp(466,16) ]
   remark #15415: vectorization support: gather was generated for the variable this:  indirect access, 64bit indexed   [ src/projector.cpp(466,54) ]
   remark #15415: vectorization support: gather was generated for the variable this:  indirect access, 64bit indexed   [ src/projector.cpp(466,89) ]
   remark #15415: vectorization support: gather was generated for the variable this:  indirect access, 64bit indexed   [ src/projector.cpp(467,16) ]
   remark #15415: vectorization support: gather was generated for the variable this:  indirect access, 64bit indexed   [ src/projector.cpp(467,54) ]
   remark #15415: vectorization support: gather was generated for the variable this:  indirect access, 64bit indexed   [ src/projector.cpp(467,89) ]
   remark #15416: vectorization support: scatter was generated for the variable f2d:  strided by 2   [ src/projector.cpp(470,5) ]
   remark #15416: vectorization support: scatter was generated for the variable f2d:  strided by 2   [ src/projector.cpp(471,5) ]
   remark #15416: vectorization support: scatter was generated for the variable f2d: masked, strided by 2   [ src/projector.cpp(474,6) ]
   remark #15305: vectorization support: vector length 8
   remark #15309: vectorization support: normalized vectorization overhead 0.111
   remark #15300: LOOP WAS VECTORIZED
   remark #15453: unmasked strided stores: 1
   remark #15458: masked indexed (or gather) loads: 12
   remark #15462: unmasked indexed (or gather) loads: 2
   remark #15475: --- begin vector loop cost summary ---
   remark #15476: scalar loop cost: 236
   remark #15477: vector loop cost: 80.250
   remark #15478: estimated potential speedup: 2.810
   remark #15487: type converts: 14
   remark #15488: --- end vector loop cost summary ---
LOOP END
```

Agenda

RELION Background

RELION ITAC and VTUE Analyze

RELION Auto-Refine Workload Optimization

RELION 2D Classification Workload Optimization

Further Optimization

Further Optimization- More efficient vectorization

1) Remove Random Access: AOS to SOA

```
FOR_ALL_DIRECT_ELEMENTS_IN_MULTIDIMARRAY(Mresol_fine) {
..
        DOUBLE diff_real = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).real - (*(Fimg_shift + n)).real;
        DOUBLE diff_imag = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).imag - (*(Fimg_shift + n)).imag;
...
         }
```

Change xx[n].real xx[n].imag to xx_real[n] and xx_imag[n]

```
        Diff_real[n] = Frefctf_real[n] – Frimg_shift_real[n];
        Diff_imag[n] = Frefcft_image[n] – Firmg_shift_imagm[n];
```

2) Use Array Reduction to easy vectorization (Intel compiler 2017)

Optimization Summary

Compile application, add these options: “-g –qopt-report=5”

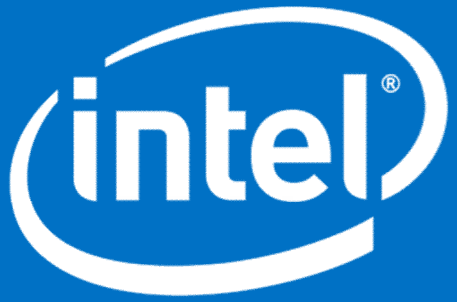
RUN Vtune Analyzer to find top hotspots

Read the xx.optrpt to find the optimization suggestion

Vectorize the loop

Try to remove random memory access

Memory alignment


intel

Legal Disclaimers

Intel, Xeon, Intel Xeon Phi, Pentium, Cilk, VTune and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

*Other names and brands may be claimed as the property of others.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel.

Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804