

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 REgularised LIkelihood OptimisatioN, 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

🚯 Applications Flaces System 🍘 🛞 🗾 👘 👔 👘 👔	lyr 7, 01194 res
S D ≥ > S Φ G Ø Waterre reveit p…x Modulared Hotenste: Hotenste Viewnight (channe) @ Intel VTime	3 mm// m/ 2 7 7 1
Reference introductor intervence interven	
Construction 14.4469 Instruction 14.4469 Instruction </th <th></th>	
Top Hotspots This section lists the mest active functions in your application, Costimizing these hotspot functions holds for improving overall application performance.	
The laborate for first carbon reaction of the laborate control of period and an implying outful appendix definition of the laborate control of the lab	Image: Constraint of the state of
C. Pallastian and Blatform Infa	
	O'U Tine
	(intel)

RELION different workload VTune Analyze

3D Auto Refine hotspots and 2D Classification hotspots

Basic Hotspots Hotspots by CPU Usage viewpoint (change) 🗇

🛛 📰 Collection Log 😝 Analysis Target 🛕 Analysis Type	e 🛍 Summary	🙈 Bottom-up 🚺 😪 Ca	ler/Callee 💰 Top-dow				
Function / Call Stack							
			CPU Ti				
Function / Call Stack		Eff	ective Time by Utilizatio				
	Idle	Poor-	OK				
MultidimArray <double>::resize</double>	0s	1256.050s	💼 💿 💹 Basic Hotspots 🛛 Hotspots by	CPU Usage viewpoint (change)	0		
MlOptimiser::storeWeightedSums	0s	423.531s					
MIOptimiser::getAllSquaredDifferences	0s	241.902s	🔐 🔍 📰 Collection Log 😌 Analysis Target 👘 Ai	nalysis Type 🛛 🛍 Summary 🚺 Bottom-up	Caller/Cal	lee 🍳	Top-down
Projector::project	0s	236.368s	Grouping: Function / Call Stack				
MIOptimiser::convertAllSquaredDifferencesToWeights	0s	103.209s					
BackProjector::backproject	0s	101.113s	0:	CPU Time+			~
operator+=	0s	96.878s	0: Eunction / Call Stack	Effective Time by Utilization	Spin Tir	ne 🕅	Overh 🕅
operator+	0s	93.073s	0:	Ellective time by othization		Others	Other
Complex::operator*	0s	88.736s	0:	🔲 Idle 📕 Poor 📋 Ok 📕 Ideal 📕 Over	Com	Other	Other
Complex::operator-	0s	47.042s	O MIOptimiser::storeWeightedSums	3514.117s	0s 8	8.276s	0s
Complex::Complex	0s	46.088s	○ Projector::rotate2D	1013.803s	Os	0s	0s
MultidimArray <double>::initConstant</double>	0s	42.235s	○ MIOptimiser::getAllSquaredDifferences	765.481s	Os	0s	0s
norm	05	37.8295	○ ▷ BackProjector::backrotate2D	324.331s	Os	0s	0s
			Complex::operator*	256.384s	0s	0s	0s
			◊ operator+	208.809s	0s	0s	0s
			Complex::operator-	204.773s	Os	0s	0s
			◊ operator+=	186.201s	0s	0s	0s
			Complex::Complex	170.552s	0s	0s	0s
			◊ operator*	58.455s	0s	0s	0s
			Complex::operator*=	56 758	0s	0s	0s
						(int	el)

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
        }
       void operator delete(void *p)
505
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 t = 0; t < Ndim; t++)
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	<pre>new data[l*ZYXdim + k*YXdim+i*Xdim+j] = 0;</pre>
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	<pre>val = DIRECT A3D ELEM(*this, k, i, j);</pre>
964	<pre>new_data[l*ZYXdim + k*YXdim+i*Xdim+j] = val;*/</pre>
965	<pre>new_data[l*ZYXdim + k*YXdim+i*Xdim+j] = DIRECT_A3D_ELEM(*this, k, i,</pre>
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	<pre>if (k >= ZSIZE(*this))</pre>
977	val = 0;
978	else if (i >= YSIZE(*this))
979	val = 0;
980	else if (j >= XSIZE(*this))
981	val = 0;
982	else
983	<pre>val = DIRECT_A3D_ELEM(*this, k, i, j);</pre>
984	<pre>new_data[l*ZYXdim + k*YXdim+i*Xdim+j] = val;</pre>
985	}
986	}

Original code

j); 944

for (long int 1 = 0; 1 < Ndim; 1++)
for (long int $k = 0$; $k < Zdim$; $k++$)
for (long int i = 0; i < Ydim; i++)
<pre>for (long int j = 0; j < Xdim; j++)</pre>
{
T val;
if (k >= ZSIZE(*this))
val = 0;
<pre>else if (i >= YSIZE(*this))</pre>
val = 0;
<pre>else if (j >= XSIZE(*this))</pre>
val = 0;
else
<pre>val = DIRECT A3D ELEM(*this, k, i, j);</pre>
<pre>new data[l*ZYXdim + k*YXdim+i*Xdim+j] = val;</pre>
}

Intel Vtune Analyze Compare(Before/After Optimize)

Source	Assembly 🔄 🔄 🗠 🤣 👰 🕤 🔍 Assembly grouping: Address						
		CPU Time:	Total	CPU Time: 5	ielf		
Source	Source	Effective Time by Utilizatio	n 🗵 Spin Ti 🗷 O.	Effective Time by Utilizatio	n 🖾 Spin		
Line		I idle Poor Ok Ideal	ver Co. Oth Oth	h I Idle Poor Ok I Ideal O	ver Co.		
1	}						
2	else						
3	<pre>new_data = new T [NZYXdim];</pre>	0.0%	0.0% 0.0% 0.0	0% Os	Os		
4	}						
5	catch (std::bad_alloc &)						
6	{						
7	REPORT_ERROR("Allocate: No space left");						
8	}						
9							
0	// Copy needed elements, fill with θ if necessary				itidim %		
11	for (long int L = 0; L < Ndim; L++)						
32	for (long int $K = 0$; $K < Zdim; K++$)					CPU Time: Total	CPU Time: Self
13	for (long int $1 = 0; 1 < \text{Yolm}; 1++)$	0.001	0.007 0.007 0.0	0.050+	Effective Time b	ry Utilization B Spin Ti B O, B	Effective Time by Utilization
14	for (long int) = 0;) < kaim;)++)	0.0%	0.0% 0.0% 0.0	1% 0.0505	Usidie Poor Ok	ideal Over Co. On. On. Jide	Poor Ok Dideal Over
5	{ 						
7	V81; if (h == 70775(#this))	0.1%	0.0% 0.0% 0.0	2 240-1	~		
/ •	11 (K > 25122(-(115)))	0.170	0.0% 0.0% 0.0	2.5405	05		
0	vac = 0; alca if (i >= VCTTE(#thicl))	0.1%	0.0% 0.0% 0.0	1% 2.726	05		
, ,	val = A:	0.176	0.070 0.070 0.0	2.7203	03		
1	also if $(i \ge YST7E(*+hic))$	0.5%	0.0% 0.0% 0.0	15 947	05		
12	val = A.	01070					
13	elce						
4	<pre>val = DIRECT A3D ELEM(*this, k, i, i);</pre>	34.7%	0.0% 0.0% 0.0	0% 1105.963s	05		
15	<pre>new data[l*ZYXdim + k*YXdim+i*Xdim+j] = val;</pre>	4.0%	0.0% 0.0% 0.0	0% 128.966s	05		
6	}						
		47 for (long int l = 0;	l < Ndim; l++)				
		48 for (long int k =	0; k < Zdim; k++)				
		49 for (long int 50 #pragma simd	1 = 0; 1 < Ydim; 1++) {				
		51 for (long	int $j = \theta$; $j < Xdim; j \leftrightarrow j$		0.2%	0.0% 0.0% 0.0% 5.22	6s 📕
		52 {		11 11 - A-	5 mil		
		54 }	acal c.stvore + K.tvore+1.vor	[s+]] = 0;	2.67	0.0% 0.0% 0.0% 1.9%77	03
		55 }					
		56 for (long int l = 0; 57 for (long int k =	<pre>l < Ndim; l++) 0: k < ZSIZE(*this); k++)</pre>				
		58 fer (long int	1 = 0: 1 < YSIZE(*this): 1+	++) [0.0%	0.0% 0.0% 0.0% 0.01	05
		59 #pragma simd	int i = 0, i < VETR(sthis)	. (++)	0.001	0.0% 0.0% 0.0% 0.14	741
		61 {	THE 1 = 0, 1 < YRITE(-(HIR)	1, 3***	0.0%	0.0% 0.0% 0.0% 0.14	(a)
		62 /*T V	al;				
		63 val =	DIRECT_A3D_ELEM(*this, k, i atal)*7VXdim + k*VXdim+**Vdi	L]): imit] = val-#/			
		65 new_d	ata[l*ZYXdim + k*YXdim+1*Xdi	[m+j] = DIRECT_A3D_ELEM(*this, k, i, j	0.1%	0.0% 0.0% 0.0% 2.10	75
		66 }					
		6/ }					



RELION Optimization(3) -- inline function

Inline function to make it be vectorized,

#vi src/ml_optimizer.optrpt <u>Before Vectorize</u>

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

```
<u>After Vectorize</u>
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

Source	Assembly 3 a 7 a a c Assembly any or Assembly any or Astronomy									
000.00		CRI Time Tr	tal	RI C	CBLI Time: Salf					
Source	Sturre	Cro Time to	Spin Ti 图 (PO TITLE: Self					
Line	ource	Effective Time by Utilization	Co Oth (Effective Time by	by Utilization					
2 622	tdu = r + s aroun $id = 1 $ or aroun $id = 1$ or aroun $id = s$ is much a contract of the set $du = s$	ldie Poor U Ok Dideal Ove	r co. out. e	All. Il Idle Poor II OK	Ideal Over					
3 633	statteri « "jrop_ua « grop_ua « mystate « mystate statterate,									
3.634	std::cerr << "Written Fing shift.spi and Fref.spi, Press any key to continue my origa									
3,635	std::cin >> c;									
8,636	}									
,637	<pre>pthread mutex unlock(&global mutex);</pre>									
638										
639	#endif									
,640	#ifdef TIMING									
,641	// Only time one thread, as I also only time one MPI process									
,642	<pre>if (my_ori_particle == exp_my_first_ori_particle)</pre>									
643	timer.tic(TIMING_DIFF_DIFF2);									
,644	FENDLT DANDIE AIFF2.									
646	if ((iter == 1.55 do firstiter cr) do always cr)	0.0%	0.0% 0.0% 0	1.050						
647			0.076 0.076 0	1.0003						
648	// Do not calculate squared-differences, but signal product									
649	// Negative values because smaller is worse in this case									
650	diff2 = 0.;									
651	DOUBLE suma2 = 0.;	0.0%	0.0% 0.0% 0	0.0% 0.080s						
,652	FOR_ALL_DIRECT_ELEMENTS_IN_WULTIDIMARRAY(Frefctf)	0.6%	0.0% 0.0% 0	0.0% 18.880s						
.653	(
8,654	<pre>diff2 -= (DIRECT_MULTIDIM_ELEM(Frefctf, n)).real * (*(Fing_shift + n)).real;</pre>	0.6%	0.0% 0.0% 0	0.0% 17.740s	(mail)					
,655	<pre>diff2 -= (DIRECT_MULTIDIM_ELEM(Frefctf, n)).imag * (*(Fing_shift + n)).imag;</pre>	1.4%	0.0% 0.0% 0	0.0% 45.879s	Igrat	product				
.000	SUMB2 += NOTM(DIRELT MULTIDIM_ELEM(FretCtT, n));	1.4%	0.0% 0.0% 0	1.0% 45.2095	this	case				
659	r // Normalized cross-correlation coefficient, divide by nover of reference (nover of i									
659	diff2 /= surf(sum2) * exploral surfxi2[inart]:	0.1%	0.0% 0.0% 0	3 6905			0.0%	0.0% 0	0.0% 0.0%	16
650	}									
		2 653			Com	ex toppy				
		3,833			C URP	ILL STREET PLOTETY THE SECTION IN	A. 100	a.au a		
		3,034			run	ALL BINELT ELEMENTS IN POLITIDIAMMATT	0.279	0.0% 0.	.076 0.070	.0
		3,655			{					
		3,656				diff2 -= (DIRECT MULTIDEM ELEM(Fr	0.7%	0.0% 0	.0% 0.0%	16
		3,657				diff2 -= (DIRECT MULTIDIM ELEM(F/	0.1%	0.0% 0	0% 0.0%	16
		3.658				000 = & DIRECT MULTIDIM FLEN(Fre				
		073 5				constant ann seas à ann seas à	0.5%	0.0% 0		
		3,039				senar += opproreac + opproreac +	0.3%	0.0% 0.	/e.e. 0.0%	4
		5,600				//sumaz ++ norm(uikeci_MULTIDIM_E				
		3,661			}					
		3,662 //jessie add end								
		3.663 /*								
		3.664		FOR ALL DIRECT ELEMENTS I	TH MULTIOTHARRAY (Erefrif)					
		2,007		/	an_macrosoft(risist)					
		5,003		1	manual dist in the					
		3,666		diff2 -= (DIRECT_MULT	TIDIM_ELEM(Frefctf, n)).real	<pre>* (*(Fing_shift + n)).real;</pre>				
		3,667		diff2 -= (DIRECT_MULT	TIDIM_ELEM(Frefctf, n)).imag	* (*(Fing_shift + n)).imag;				
		3,668		suma2 ++ norm(DIRECT	MULTIDIM ELEM(Frefctf, n));					
		3.669		}						

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

Basic Hotspots Hotspots by CPU Usage viewpoint (change)									
< 📟 Collection Log 😝 Analysis Target 🔺 Analysis Type 🕅 Summary 😪	Bottom-up 🔗 Caller/Callee 😽 Top-down Tre	e 🔣	Platfo	orm					
	CPU Time: Total			~	CPU				
Function	Effective Time by Utilization	Spin T	ri ⊠	0.	Effective Time by Utili:				
	🛛 Idle 📕 Poor 📋 Ok 📳 Ideal 📕 Over	Co.	Oth	Oth	🛛 Idle 📕 Poor 📋 Ok 🔳 Ideal 🚺				
MIOptimiser::doThreadExpectationSomeParticles	99.0%	0.0%	0.1%	0.0%	0s				
MIOptimiser::expectationOneParticle	99.0%	0.0%	0.1%	0.0%	Os				
MlOptimiser::storeWeightedSums	67.5%	0.0%	0.1%	0.0%	3514.117s				
MIOptimiser::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	<pre>// Suggestion Robert Sinkovitz: merge difference and scale steps to make bet</pre>					
4,610	FOR ALL DIRECT ELEMENTS IN MULTIDIMARRAY(Mresol fine)	2.7%	0.0%	0.0%	0.0%	106.3195
4,611	{					
4,612	<pre>int ires = DIRECT_MULTIDIM_ELEM(Mresol_fine, n);</pre>	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	<pre>// Use FT of masked image for noise estimation!</pre>					
4,616	DOUBLE diff_real = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).real - (*(Fing	2.9%	0.0%	0.0%	0.0%	113.965s
4,617	DOUBLE diff_inag = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).inag - (*(Fing	0.8%	0.0%	0.0%	0.0%	31.006s
4,618	DOUBLE wdiff2 = weight * (diff_real*diff_inag*diff_inag)	1.8%	0.0%	0.0%	0.0%	70.536s
4,619	// group-wise sigma2 noise					
4.620	<pre>DIRECT_MULTIDIM_ELEM(thr_wsum_sigma2_noise[group_id], ires) += wdiff2</pre>	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 SUNXA					
4,626	sunXA = (DIR	0.1%	0.0%	0.0%	0.0%	2.280s
4,627	sunXA += (DI	0.2%	0.0%	0.0%	0.0%	6.767s
4,628	DIRECT_AID_E	1.2%	0.0%	0.0%	0.0%	48.005s
4,629	sunA2 = (DIR	0.0%	0.0%	0.0%	0.0%	1.468s
4,630	sunA2 += (DI	0.4%	0.0%	0.0%	0.0%	14.967s
4,631	DIRECT_AID_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	<pre>#pragma simd reduction(+:inner_exp_wsum_norm_correction)</pre>					
4,642	for (long int Mresol_initer=0; Mresol_initer <mresol_inner_size; mresol_initer++)="" td="" {<=""><td>1.4% 📒</td><td>0.0%</td><td>0.0%</td><td>0.0.</td><td>40.074s 📒</td></mresol_inner_size;>	1.4% 📒	0.0%	0.0%	0.0.	40.074s 📒
4,643	<pre>long int n = Mresol_outiter*Mresol_inner_size + Mresol_initer;</pre>					
4,644	<pre>int ires = DIRECT_MULTIDIM_ELEM(Mresol_fine, n);</pre>	0.0%	0.0%	0.0%	0.0.	0.350s
4,645	if (ires > -1)	0.7% 📒	0.0%	0.0%	0.0.	20.269s
4,646	{					
4,647	<pre>// Use FT of masked image for noise estimation!</pre>					
4,648	<pre>DOUBLE diff_real = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).real - (*(Fimg_shift + n)).rea</pre>	7.1%	0.0%	0.0%	0.0.	204.495s
4,649	DOUBLE diff_imag = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).imag - (*(Fimg_shift + n)).ima	0.2%	0.0%	0.0%	0.0.	4.621s
4,650	<pre>Mresol_fine_wdiff2[Mresol_initer] = weight * (diff_real*diff_real + diff_imag*diff_i</pre>	2.2%	0.0%	0.0%	0.0.	63.237s
4,651	// For norm_correction					
4,652	<pre>//exp_wsum_norm_correction[ipart] += Mresol_fine_wdiff2[Mresol_initer];</pre>					
4,653	<pre>inner_exp_wsum_norm_correction += Mresol_fine_wdiff2[Mresol_initer];</pre>	0.3%	0.0%	0.0%	0.0.	8.524s
4,654	}					
4,655	}					
4,656	#pragma ivdep					
4,657	<pre>for (long int Mresol_initer=0; Mresol_initer<mresol_inner_size; mresol_initer++)="" pre="" {<=""></mresol_inner_size;></pre>					
4,658	<pre>long int n = Mresol_outiter*Mresol_inner_size + Mresol_initer;</pre>					
4,659	<pre>int ires = DIRECT_MULTIDIM_ELEM(Mresol_fine, n);</pre>					
4,660	<pre>//int ires1 = DIRECT_MULTIDIM_ELEM(Mresol_fine, n+1);</pre>					
4,661	<pre>//_mm_prefetch((char *)(&DIRECT_A1D_ELEM(mymodel.data_vs_prior_class[exp_iclass], ires1)</pre>					
4,662	if (ires > -1)	0.3%	0.0%	0.0%	0.0.	7.929s
4,663	{					
4,664	// group-wise sigma2_noise					
4,665	<pre>//DIRECT_MULTIDIM_ELEM(thr_wsum_sigma2_noise[group_id], ires) += Mresol_fine_wdiff2[</pre>					
4,666	<pre>Mresol_fine_sigma2_noise[n] += Mresol_fine_wdiff2[Mresol_initer];</pre>	1.1% 📒	0.0%	0.0%	0.0.	32.277s 📒
4,667	}					
4,668	}					
4,669						
4,670	<pre>}//end for Mresol outier first loop</pre>					



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< th=""><th>1.6%</th><th>0.0%</th><th>0.0%</th><th>0.0.</th><th>44.698s</th></mresol_fi<>	1.6%	0.0%	0.0%	0.0.	44.698s
4,698	<pre>int n = Mresol_fine_data_vs_prior_class[Mresol_fine_data_vs_prior_class_idx];</pre>	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	<pre>sumXA = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).real * (*(Fimg_shift + n)).real;</pre>	1.2%	0.0%	0.0%	0.0.	35.711s 📒
4,702	<pre>sumXA += (DIRECT_MULTIDIM_ELEM(Frefctf, n)).imag * (*(Fimg_shift + n)).imag;</pre>	0.3%	0.0%	0.0%	0.0.	7.493s 🛛
4,703	<pre>DIRECT_A1D_ELEM(exp_wsum_scale_correction_XA[ipart], ires) += weight * sumXA;</pre>	3.8%	0.0%	0.0%	0.0.	108.124s
4,704	<pre>sumA2 = (DIRECT_MULTIDIM_ELEM(Frefctf, n)).real * (DIRECT_MULTIDIM_ELEM(Frefctf,</pre>	0.1%	0.0%	0.0%	0.0.	2.181s
4,705	<pre>sumA2 += (DIRECT_MULTIDIM_ELEM(Frefctf, n)).imag * (DIRECT_MULTIDIM_ELEM(Frefctf</pre>	1.0% 📒	0.0%	0.0%	0.0.	30.138s 📒
4,706	<pre>DIRECT_A1D_ELEM(exp_wsum_scale_correction_AA[ipart], ires) += weight * sumA2;</pre>	0.9% 📒	0.0%	0.0%	0.0.	26.390s 📕
4,707	}					
4 700						

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 Fing 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

Get hotspots info from Vtune analyze

MIOptimiser::expectationOneParticle	99.0%	0.0%	0.1%	0.0%	Os
MIOptimiser::storeWeightedSums	67.5%	0.0%	0.1%	0.0%	3514.117s
MIOptimiser::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

Line 410: For loop is not vectorized

T-maximum da		CPU Time: Total		CPU Time: Self	
Line -	Source	Effective Time by Utilization	図 Spin Ti 团 O. 团 Co. Oth Oth	Effective Time by Utilization	Spin T Co.
23	if (xp < 0)	0.4%	0.0% 0.0% 0.0%	26.118s	Os
24	(
25	<pre>// Get complex conjugated hermitian symmetry pair</pre>				
26	xp = -xp;	0.0%	0.0% 0.0% 0.0%	2.8095	05
27	yp = -yp;	0.5%	0.0% 0.0% 0.0%	35.060s	Os
28	is_neg_x - true;				_
29	3				
30	else				
31					
32	is_neg_x = Talse;				
33	1				
34	// Tellinger interestation (with physical engels)				
36	// fultimed interpolation with physical coords/				
37	// To that you use DIPECT ADD BLOW rather than ADD BLOW				
30	y = ELOP(y).	0.7%	0.0% 0.0% 0.0%	52 5420	Or
10	$f_{\rm r} = r_{\rm r} = r_{\rm r}$	0.3%	0.0% 0.0% 0.0%	24 318	0s
40	$x_1 = x_2 + x_3$	0.070	0.070 0.070 0.070	24.5103	0.5
41					
42	v0 = FL00R(vp):	1.1%	0.0% 0.0% 0.0%	82.8035	Os
43	$fy = yp - y\theta$:				
44	y0 -= STARTINGY(data);	0.1%	0.0% 0.0% 0.0%	9.864s	Os
45	y1 = y0 + 1;	0.1%	0.0% 0.0% 0.0%	5.493s	Os
46		7.0			
47	// Matrix access can be accelerated through pre-calculation of z0*xydim etc.				
48	d00 = DIRECT_A2D_ELEM(data, y0, x0);	1.8%	0.0% 0.0% 0.0%	133.809s	Os
49	<pre>d01 = DIRECT_A2D_ELEM(data, y0, x1);</pre>	0.5%	0.0% 0.0% 0.0%	35.850s	Os
50	d10 = DIRECT_A2D_ELEM(data, y1, x0);	1.4%	0.0% 0.0% 0.0%	104.048s	Os
51	dl1 = DIRECT_A2D_ELEM(data, y1, x1);	0.5%	0.0% 0.0% 0.0%	36.330s 🚾	Os
52					
53	// Set the interpolated value in the 2D output array				
54	dx0 = LIN_INTERP(fx, d00, d01);	4.7%	0.0% 0.0% 0.0%	75.397s	Os
55	dx1 = LIN_INTERP(fx, d10, d11);	4.7%	0.0% 0.0% 0.0%	75.496s	Os
56	DIRECT_A2D_ELEM(f2d, i, x) = LIN_INTERP(fy, dx0, dx1);	5.3%	0.0% 0.0% 0.0%	137.2525	Os
57	// Take complex conjugated for half with negative x				
80	1T (1s_neg_x)	1.10	0.001 0.001 0.001	10 202-	
59	DIRECT_AZD_ELEM(TZG, 1, x) = CONJ(DIRECT_AZD_ELEM(TZG, 1, x));	1.1%	0.0% 0.0% 0.0%	40.301s	OS
0	} // endit ikilineak				
	PISP IT LIDTPEDDIATOR AN DEAREST DELIVERIDE J				



410	for (int x=0; x <= my r_max; x++) 44	142 y0 = FLOOR(yp);
411	£ 44	fy = yp - y0;
412	<pre>// Only include points with radius < max_r (exclude points outside circle in square) 44</pre>	<pre>y0 -= STARTINGY(data);</pre>
413	r2 = x * x + y2; 44	$y_1 = y_0 + 1;$
414	if (r2 > max_r2) 44	146
415	continue; 44	447 // Matrix access can be accelerated through pre-calculation of z0*xydim etc.
416	44	d00 = DIRECT A2D ELEM(data, y0, x0);
417	// Get logical coordinates in the 3D map 44	d01 = DIRECT A2D ELEM(data, y0, x1);
418	xp = Ainv(0,0) * x + Ainv(0,1) * y; 45	d10 = DIRECT A2D ELEM(data, v1, x0);
419	yp = Ainv(1,0) * x + Ainv(1,1) * y;	d11 = DIRECT A2D ELEM(data, v1, x1):
420	if (interpolator == TRILINEAR r2 < min_r2_nn) 45	152
421	{	// Set the interpolated value in the 2D output array
422	// Only asymmetric half is stored 45	dx0 = LIN INTERP(fx, d00, d01):
423	if (xp < 0) 45	dx1 = LIN INTERP(fx, d10, d11):
424	45	DIRECT A2D FLEW(f2d, 1, x) = LIN INTERP(fv, dx0, dx1):
425	// Get complex conjugated hermitian symmetry pair	// Take complex conjugated for half with pegative x
426	xp = -xp;	17) The compagate to main with negative a
427	yp = -yp;	$ \frac{1}{12} $
428	is_neg_x = true;	1// and f TELLINEAL
429)	(1) algo if (interpolation NEADERT NEICUBOUD)
430	else	(interpolator - REARST AFIGHOUR)
431	{	
432	is_neg_x = false;	xu = KOUNU(XD);
433)	$y_0 = k(0) k(0) (y_0);$
434	10	11 (XU < 0)
435	// Trilinear interpolation (with physical coords)	DIRECT_A2D_ELEM(f2d, 1, x) = conj(A2D_ELEM(data, -y0, -x0));
436	// Subtract STARTINGY to accelerate access to data (STARTINGX=0) 46	ler else
437	// In that way use DIRECT_A3D_ELEM, rather than A3D_ELEM 46	DIRECT_A2D_ELEM(f2d, i, x) = A2D_ELEM(data, y0, x0);
438	$x_0 = FLOOR(x_p);$	<pre>609 } // endif NEAREST_NEIGHBOUR</pre>
439	fx = xp - x0; 47	else else
440	x1 = x0 + 1; 47	<pre>REPORT_ERROR("Unrecognized interpolator in Projector::project");</pre>
441	45	172 l (/ and if y_loop

LOOP BEGIN at src/projector.cpp(<mark>410</mark>,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++){ ...}</pre>
```

```
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++){}
}</pre>
```

```
}
```



}

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





Legal Disclaimers

Copyright © 2014 Intel Corporation

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