Parallel Progrmg: Sci & Engrg

Machine Problem 2

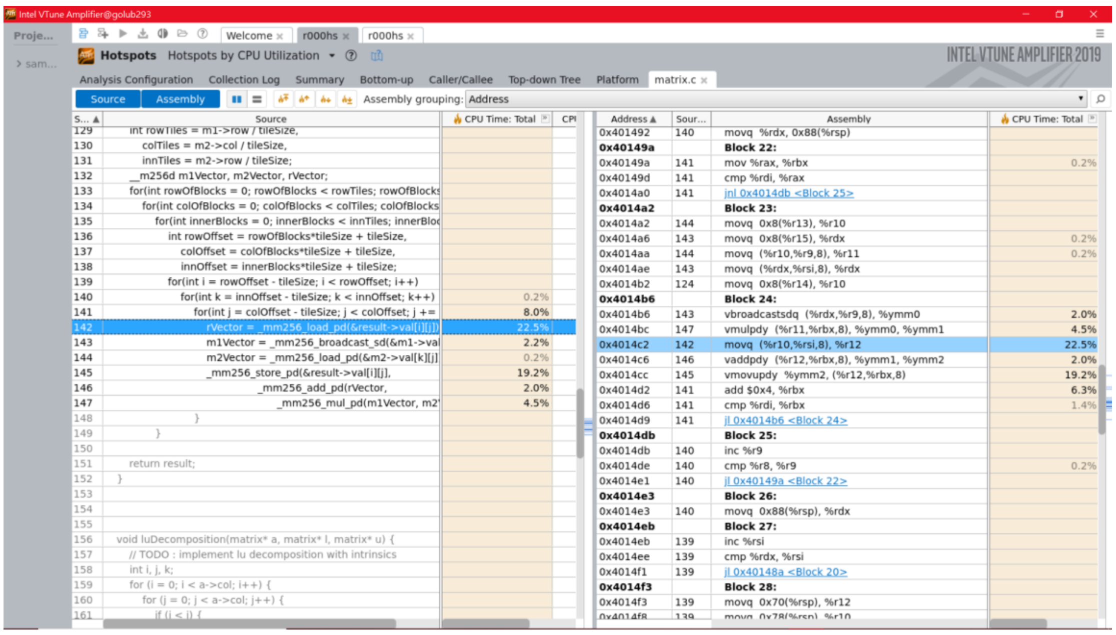
yhyuan2

A. Q: For the version of matmul without vector intrinsics, did the compiler generate packed vector instructions? If so, pick one of the packed vector instructions it generated and briefly explain what it does. Hint: Consult the assembly tab.

A: Compiler did generate packed vector instructions. Take vmulpd %ymm15, %ymm10, %ymm10 for example. It multiply %ymm10 with itself and put the result into %ymm15.

B. Q: Which of the vector intrinsics in your hand-vectorized program dominated your program's execution time? Briefly explain why and include a screenshot of your hand-vectorized program's top hotspot like Figure 2 in your report.

A: rVector = _mm256_load_pd(&result->val[i][j]) dominates the execution time because it accesses result value from memory. Accessing values from memory will need to retrieve data and place them in cache to execute. So it will spend more time.



Assembly grouping: Address

S...	Source	CPU Time: Total	CPU
129	int rowTiles = m1->row / tileSize,		
130	colTiles = m2->col / tileSize,		
131	innTiles = m2->row / tileSize;		
132	__m256d m1Vector, m2Vector, rVector;		
133	for(int rowOfBlocks = 0; rowOfBlocks < rowTiles; rowOfBlocks		
134	for(int colOfBlocks = 0; colOfBlocks < colTiles; colOfBlocks		
135	for(int innerBlocks = 0; innerBlocks < innTiles; innerBloc		
136	int rowOffset = rowOfBlocks*tileSize + tileSize,		
137	colOffset = colOfBlocks*tileSize + tileSize,		
138	innOffset = innerBlocks*tileSize + tileSize;		
139	for(int i = rowOffset - tileSize; i < rowOffset; i++)		
140	for(int k = innOffset - tileSize; k < innOffset; k++)	0.2%	
141	for(int j = colOffset - tileSize; j < colOffset; j +=	8.0%	
142	rVector = _mm256_load_pd(&result->val[i][j])	22.5%	
143	m1Vector = _mm256_broadcast_sd(&m1->val	2.2%	
144	m2Vector = _mm256_load_pd(&m2->val[k][j]	0.2%	
145	_mm256_store_pd(&result->val[i][j],	19.2%	
146	_mm256_add_pd(rVector,	2.0%	
147	_mm256_mul_pd(m1Vector, m2	4.5%	
148	}		
149	}		
150			
151	return result;		
152	}		
153			
154			
155			
156	void luDecomposition(matrix* a, matrix* l, matrix* u) {		
157	// TODO : implement lu decomposition with intrinsics		
158	int i, j, k;		
159	for (i = 0; i < a->col; i++) {		
160	for (j = 0; j < a->col; j++) {		
161	if (i < j) {		

Address	Sour...	Assembly	CPU Time: Total
0x401492	140	movq %rdx, 0x88(%rsp)	
0x40149a		**Block 22:**	
0x40149a	141	mov %rax, %rbx	0.2%
0x40149d	141	cmp %rdi, %rax	
0x4014a0	141	jnl 0x4014db <Block 25>	
0x4014a2		**Block 23:**	
0x4014a2	144	movq 0x8(%r13), %r10	
0x4014a6	143	movq 0x8(%r15), %rdx	0.2%
0x4014aa	144	movq (%r10,%r9,8), %r11	0.2%
0x4014ae	143	movq (%rdx,%rsi,8), %rdx	
0x4014b2	124	movq 0x8(%r14), %r10	
0x4014b6		**Block 24:**	
0x4014b6	143	vbroadcastsdq (%rdx,%r9,8), %ymm0	2.0%
0x4014bc	147	vmulpdy (%r11,%rbx,8), %ymm0, %ymm1	4.5%
0x4014c2	142	movq (%r10,%rsi,8), %r12	22.5%
0x4014c6	146	vaddpdy (%r12,%rbx,8), %ymm1, %ymm2	2.0%
0x4014cc	145	vmovupdy %ymm2, (%r12,%rbx,8)	19.2%
0x4014d2	141	add $0x4, %rbx	6.3%
0x4014d6	141	cmp %rdi, %rbx	1.4%
0x4014d9	141	jl 0x4014b6 <Block 24>	
0x4014db		**Block 25:**	
0x4014db	140	inc %r9	
0x4014de	140	cmp %r8, %r9	0.2%
0x4014e1	140	jl 0x40149a <Block 22>	
0x4014e3		**Block 26:**	
0x4014e3	140	movq 0x88(%rsp), %rdx	
0x4014eb		**Block 27:**	
0x4014eb	139	inc %rsi	
0x4014ee	139	cmp %rdx, %rsi	
0x4014f1	139	jl 0x40148a <Block 20>	
0x4014f3		**Block 28:**	
0x4014f3	139	movq 0x70(%rsp), %r12	
0x4014f8	139	movq 0x78(%rsp), %r10	

C. Q: How did the performance of your hand-vectorized version compare to the original version? Include the MFLOPs rates for both versions and briefly comment about whether or not you think the compiler's auto-vectorization was adequate and why.

A: Hand-vectorized version is much faster than original version. MFLOPs rates are listed below. I think auto-vectorization isn’t adequate. Auto-vectorization is still not smart enough to find how to vectorize our code will make it faster. Also, multiplication in original version is simply row times column. It’s a little bit different from the implementation of multiplication in hand-vectorized version. That’s hard for complier to vectorize original version.

	Hand-vectorized Version	Original Version
MFLOPs	5534.283367	2633.243824