Vectorization

Silvio Stanzani , Raphael Cóbe , Rogério Iope
UNESP - Núcleo de Computação Científica
silvio@ncc.unesp.br , rmcobe@ncc.unesp.br , rogerio@ncc.unesp.br

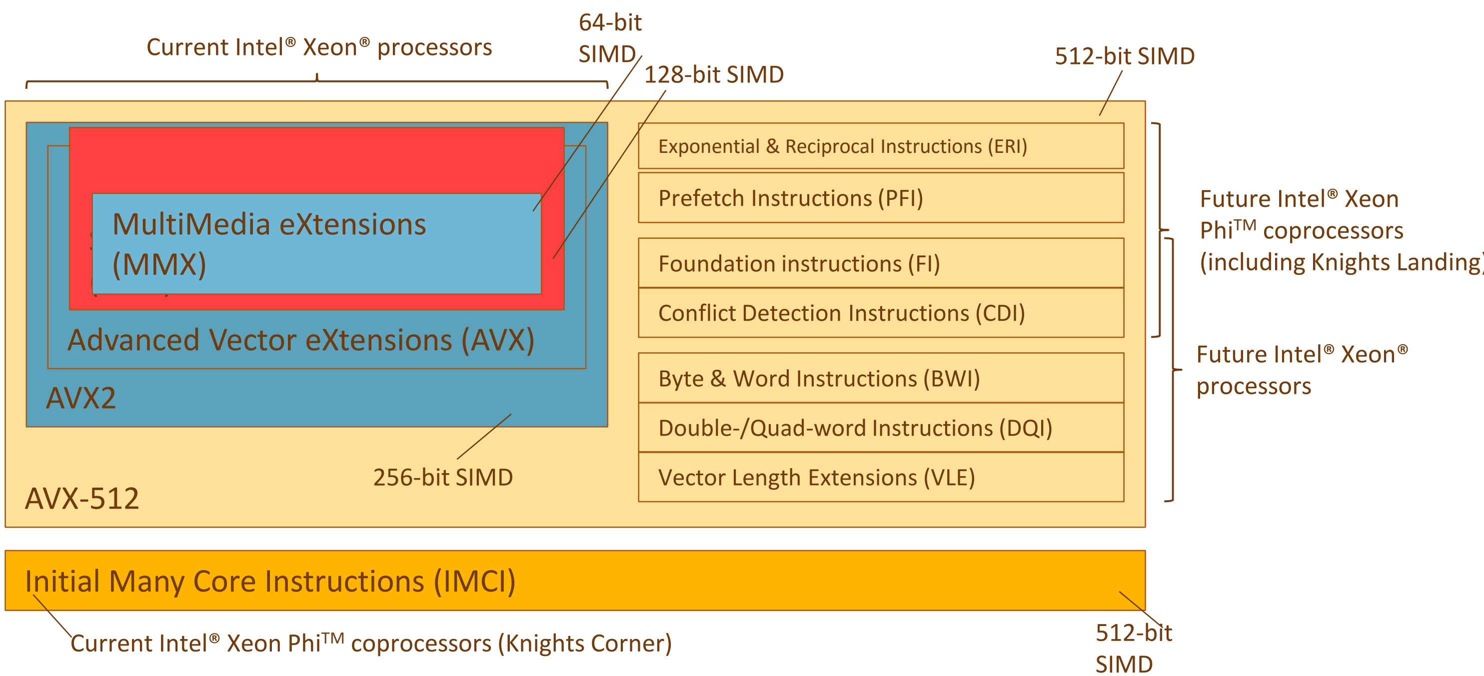
Agenda

- Vector Processing Units;
- Profiling;
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization.

Agenda

- **Vector Processing Units;**
- Profiling;
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization.

Past, present, and future of Intel SIMD types



Intel® AVX2/IMCI/AVX-512 differences

	Intel® Initial Many Core Instructions **IMCI**	Intel® Advanced Vector Extensions 2 **AVX2**	Intel® Advanced Vector Extensions 512 **AVX-512**
Introduction	2012	2013	2015
Products	Knights Corner	Haswell, Broadwell	Knights Landing, future Intel® Xeon® and Xeon® Phi™ products
Register file	SP/DP/int32/int64 data types 32 x 512-bit SIMD registers 8 x 16-bit mask registers	SP/DP/int32/int64 data types 16 x 256-bit SIMD registers No mask registers (instr. blending)	SP/DP/int32/int64 data types 32 x 512-bit SIMD registers 8 x (up to) 64-bit mask
ISA features	Not compatible with AVX*/SSE* No unaligned data support Embedded broadcast/cvt/swizzle MVEX encoding	Fully compatible with AVX/SSE* Unaligned data support (penalty) VEX encoding	Fully compatible with AVX*/SSE* Unaligned data support (penalty) Embedded broadcast/rounding EVEX encoding
Instruction features	Fused multiply-and-add (FMA) Partial gather/scatter Transcendental support	Fused multiply-and-add (FMA) Full gather	Fused multiply-and-add (FMA) Full gather/scatter Transcendental support (ERI only) Conflict detection instructions PFI/BWI/DQI/VLE (if applies)

Agenda

- Vector Processing Units;
- **Profiling;**
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization.

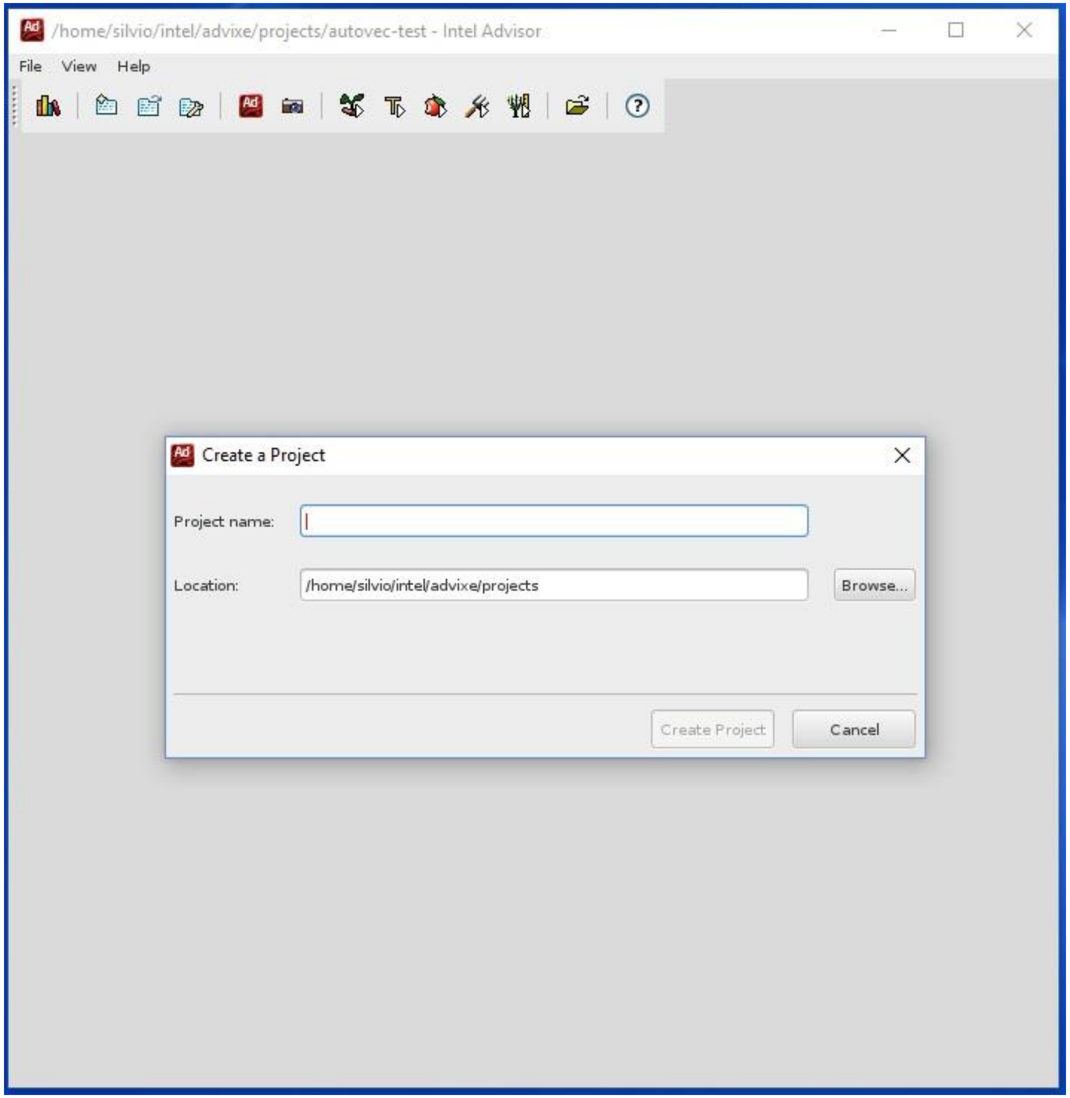
Intel Advisor

- Evaluate multi-threading parallelization
- Intel® Advisor XE
    - ❑ Performance modeling using several frameworks for multi-threading in processors and co-processors:
        - OpenMP, Intel® Cilk ™ Plus, Intel® Threading Bulding Blocks
        - C, C++, Fortran (OpenMP only) and C# (Microsoft TPL)
    - ❑ Identify parallel opportunities
        - Detailed information about vectorization;
        - Check loop dependencies;
    - ❑ Scalability prediction: amount of threads/performance gains
    - ❑ Correctness (deadlocks, race conditions)

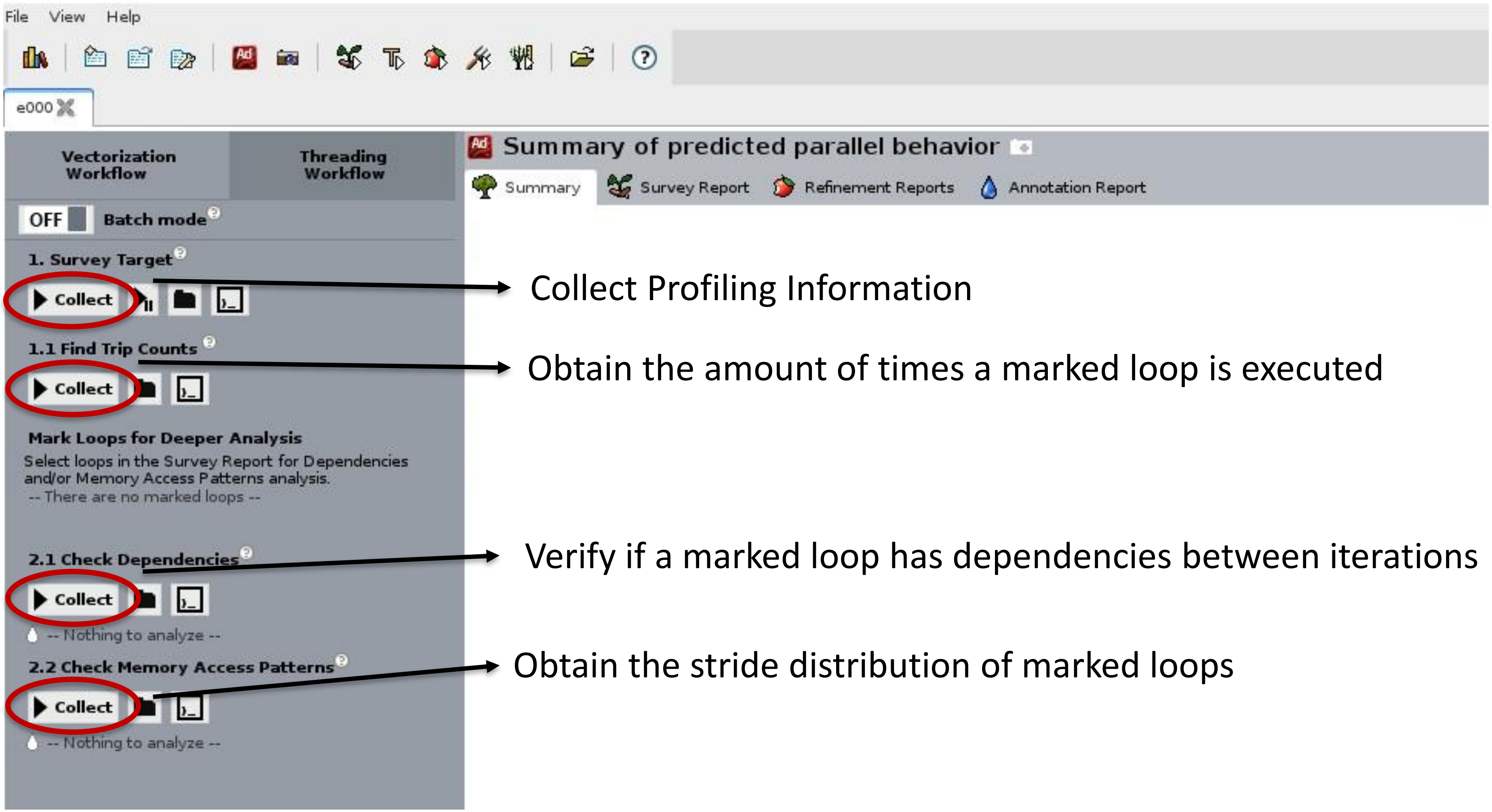
Intel Advisor

- Survey Target;
  - Vectorization of loops: detailed information about vectorization;
  - Total Time: elapsed time on each loop considering the time involved in internal loops;
  - Self Time: elapsed time on each loop not considering the time involved in internal loops;
- Find Trip Counts;
  - Analysis to identify how many time particular loops run;
- Check Dependencies;
  - Analysis it there are many loop-carried dependencies;
- Check Memory Access Patterns.
  - Analysis to identify how your code is iterating with memory.

Advisor – New Project



Advisor - Analysis



Collect Profiling Information

Obtain the amount of times a marked loop is executed

Verify if a marked loop has dependencies between iterations

Obtain the stride distribution of marked loops

Advisor – Survey Target

Summary | Survey Report | Refinement Reports | Annotation Report

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Vectorized Loops Vec...	Efficiency	Gai...	VL (Vector Length)	Compiler Estimated Gain	Instruction Set Analysis Traits
[loop in main at vec.c:50]		2 Inefficient memory access patterns present	0.730s	0.730s	Vectorized (Body)	AVX2		2.20x	16	2.20x	Extracts; Type Conversions
[loop in main at vec.c:63]		2 Data type conversions present	0.680s	0.680s	Vectorized Versions	AVX	~71%	5.71x	2; 8	<5.71x	Extracts; Inserts; Type Conversions
[loop in main at vec.c:55]		1 Data type conversions present	0.490s	0.490s	Vectorized (Body)	AVX2	~72%	5.79x	8	5.79x	Extracts; Inserts; Type Conversions

Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: vec.c:50 main

Line	Source	Total Time	%	Loop Time	%	Traits
36	`randV=0;`					
37						
38	`a = (double*)  mm malloc(msize * msize *sizeof(double),64);`					
39	`b = (double*)  mm malloc(msize * msize *sizeof(double),64);`					
40	`c = (double*)  mm malloc(msize * msize * sizeof(double),64);`					
41						
42	`srand(time(NULL));`					
43	`randV=rand();`					
44	`randV=randV*0.11;`					
45	`printf("randV %f\n", randV);`					
46						
47						
48	`for(j=0; j<10000; j++) {`			1,899.980m		
49						
50	`for(i=0; i<40000; i++) {`			729.970ms		
51	`aosobj[i].x=i + randV;`	229.961ms				Extracts; Type Conversions
52	`aosobj[i].y=i*i+ randV;`	290.000ms				Extracts; Type Conversions
53	`aosobj[i].z=i+i+ randV;`	210.009ms				Extracts; Type Conversions
54	`}`					
55	`for(i=0; i<40000; i++) {`	30.003ms		490.029ms		
56	`soaobj.x[i]=i+i+ randV;`	129.997ms				Extracts; Inserts; Type Conversions
57	`soaobj.y[i]=i-i+ randV;`	40.001ms				Type Conversions
58	`soaobj.z[i]=i*i+ randV;`	290.027ms				Extracts; Inserts; Type Conversions
59	`}`					
60	`randV=rand();`					Type Conversions
61	`randV=randV*0.11;`					
62						
63	`for(i=0; i<40000; i++) {`	9.999ms		679.985ms		
64	`aosobj[i].x=    aosobj[i].y+  aosobj[i].z + randV;`	489.987ms				Extracts; Inserts; Type Conversions
65	`}`					
66						
67	`for(i=0; i<40000; i++) {`					
68						
69	`soaobj.x[i]= soaobj.y[i]+ soaobj.z[i] + randV;`	179.999ms				Extracts; Inserts; Type Conversions
70	`}`					
71	`}`					

Advisor – Survey Target

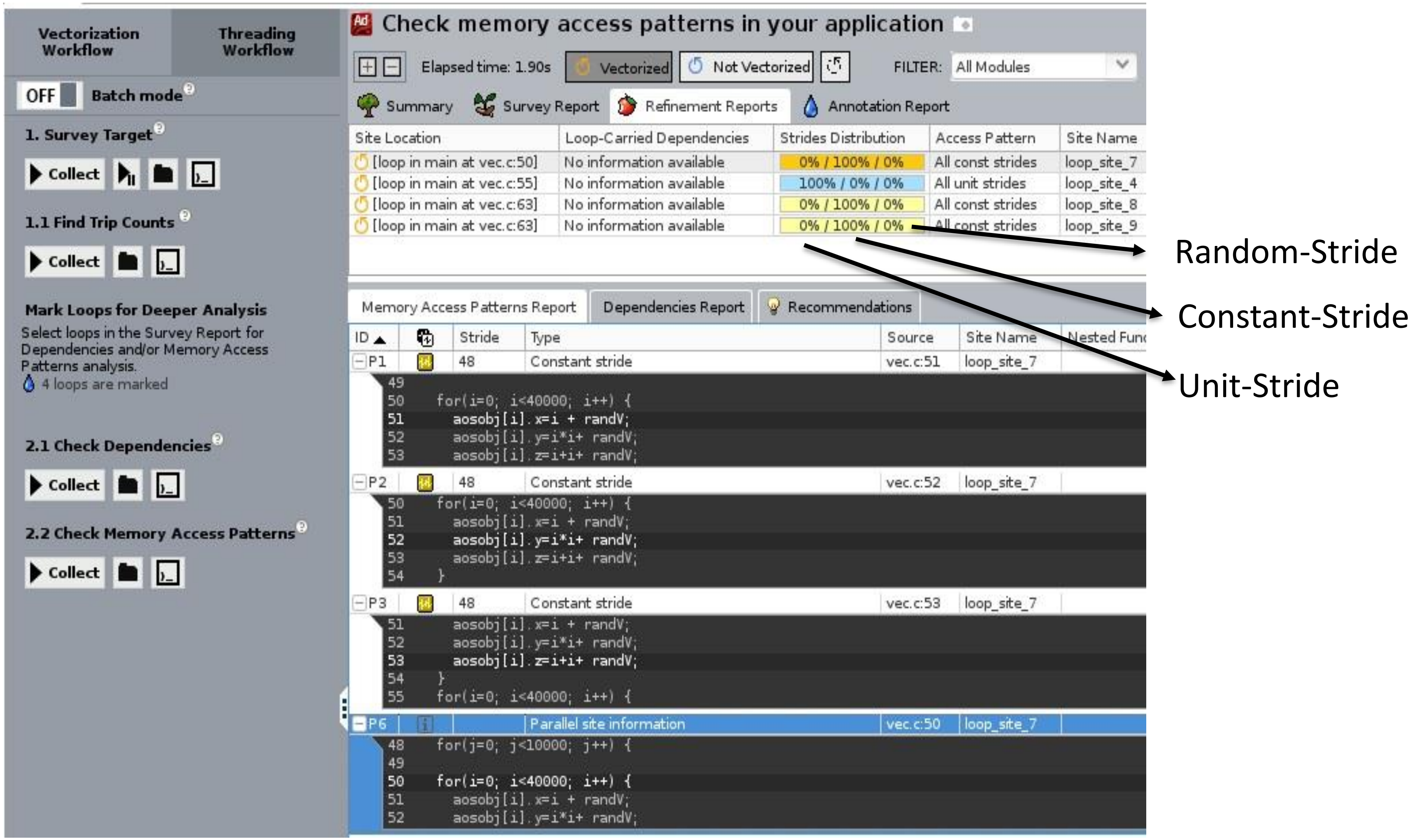
Instruction Set Analysis

Traits	Data Types	Number of Vector Registers	Vector Widths	Instruction Sets
Extracts; Type Conversions	Float32; Float64; Int32; UIn...	15	128/256	AVX; AVX2
Extracts; Inserts; Type Conversions	Float32; Float64	14; 15	128; 256	AVX
Extracts; Inserts; Type Conversions	Float32; Float64; Int32; UIn...	16	256	AVX; AVX2

Advanced

Transformations	Unroll Factor	Vectorization Details	Optimization Details	Location
				vec.c:50
Fused; Unrolled	4		LOOP WAS DISTRIBUTED, CHUNK 1; LOOP WAS DISTRIBUTED, C...	vec.c:63
				vec.c:55

Advisor – Memory Access Patterns



Agenda

- Vector Processing Units;
- Profiling;
- **Optimizing Memory Access;**
- Auto Vectorization;
- Guided Vectorization.

Stride (array elements)

- Stride:
  - Step size between consecutive access of array elements;

- Strided access with stride k means touching every kth memory element
  - Unit Stride :
    - Sequential access (0, 1, 2, 3, 4, 5, 6, ...)
  - Non-unit stride
    - Constant Stride =
      - 2 is (0, 2, 4, 6, 8, ...)
    - k is (0, k, 2k, 3k, 4k, ...)
    - Random Access;

- Strides > 1 commonly found in multidimensional data
  - Row accesses (stride=N) & diagonal accesses (stride=N+1)
  - Scientific computing (e.g., matrix multiplication)

Padding

- Data structures may have members with different sizes.
- To maintain proper alignment the translator normally inserts additional unnamed data members so that each member is properly aligned.
- Example:

```
struct stu_a {
    int i;
    char c;
};
```

- Actual size 4+1 (5)

i	i	i	i	c	i	i	i	c	i	i	i	c
8 bytes								8 bytes				

- After Padding size 4+4 (5)

i	i	i	i	c	c	c	c	i	i	i	i	c
8 bytes								8 bytes				

- …

Padding

- Vectorization more efficient with unit strides
  - Non-unit strides will generate gather/scatter
  - Unit strides also better for data locality

Demo: padd.c

Icc padd.c –o padd

./padd

Data layout

- AoS vs SoA (Array of Structures vs Structure of Arrays)
  - Layout your data as Structure of Arrays (SoA)

```
// Array of Structures (AoS)
struct coordinate {
    float x, y, z;
} crd[N];
    …
for (int i = 0; i < N; i++)
    … = … f(crd[i].x, crd[i],y, crd[i].z);
```

Consecutive elements in memory

x0	y0	z0	x1	y1	z1	…	x(n-1)	y(n-1)	z(n-1)

```
// Structure of Arrays (SoA)
struct coordinate {
    float x[N], y[N], z[N];
} crd;
    …
for (int i = 0; i < N; i++)
    … = … f(crd.x[i], crd.y[i], crd.z[i]);
```

Consecutive elements in memory

x0 x1 … x(n-1)	y0 y1 … y(n-1)	z0 z1 … z(n-1)

Data Alignment

How to…	Syntax	Semantics
…align data	`void* _mm_malloc(int size, int n)` `void* _mm_free(int size)`	Allocate memory on heap aligned to *n* byte boundary.
	`int posix_memalign (void **p, size_t n, size_t size)`	
	`__declspec(align(`*n*`))` *array*	Alignment for variable declarations.
…tell the compiler about it	`#pragma vector aligned`	Vectorize assuming all array data accessed are aligned (may cause fault otherwise).
	`__assume_aligned(`*array*`,` *n*`)`	Compiler may assume array is aligned to *n* byte boundary.

Loop Splitting

- Loop Splitting
  - Set of techniques to breaking the loop into multiple loops which have the same body, but iterate over different contiguous portions of the index range.
    - ❑ Body
    - ❑ Peel Loop: beginning of loop
    - ❑ Remainder Loop: end of loop
- Loop Unrolling
  - Execute a set of iterations as a single iteration;

Vectorization with multi-version loops

Peel loop
Alignment purposes
Might be vectorized

```
LOOP BEGIN at gas_dyn2.f90(2330,26)
<Peeled>
   remark #15389: vectorization support: reference AMAC1U has unaligned
access
   remark #15381: vectorization support: unaligned access used inside loop
body
   remark #15301: PEEL LOOP WAS VECTORIZED
LOOP END
```

Main loop
Vectorized
Unrolled by x2 or x4

```
LOOP BEGIN at gas_dyn2.f90(2330,26)
   remark #25084: Preprocess Loopnests: Moving Out Store
   remark #15388: vectorization support: reference AMAC1U has aligned access
   remark #15399: vectorization support: unroll factor set to 2
   remark #15300: LOOP WAS VECTORIZED
   remark #15475: --- begin vector loop cost summary ---
   remark #15476: scalar loop cost: 8
   remark #15477: vector loop cost: 0.620
   remark #15478: estimated potential speedup: 15.890
   remark #15479: lightweight vector operations: 5
   remark #15488: --- end vector loop cost summary ---
   remark #25018: Total number of lines prefetched=4
   remark #25019: Number of spatial prefetches=4, dist=8
   remark #25021: Number of initial-value prefetches=6
LOOP END
```

Remainder loop
Remainder iterations
Might be vectorized

```
LOOP BEGIN at gas_dyn2.f90(2330,26)
<Remainder>
   remark #15388: vectorization support: reference AMAC1U has aligned access
   remark #15388: vectorization support: reference AMAC1U has aligned access
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END
```

Agenda

- Vector Processing Units;
- Profiling;
- Optimizing Memory Access;
- **Auto Vectorization;**
- Guided Vectorization.

Vectorization on Intel® compilers

Easy of use

Auto Vectorization

- Compiler knobs

Guided Vectorization

- Compiler hints/pragmas
- Array notation
- Elemental Functions

Low level Vectorization

- C/C++ vector classes
- Intrinsics/Assembly

Fine control

Auto vectorization

- Relies on the compiler for vectorization
  - No source code changes
  - Enabled with `-vec` compiler knob (default in `-O2` and `-O3` modes)
- Compiler smart enough to apply loop transformations
  - It will allow to vectorize more loops

Option	Description
`-O0`	Disables all optimizations.
`-O1`	Enables optimizations for speed which are know to not cause code size increase.
`-O2/-O` (default)	Enables intra-file interprocedural optimizations for speed, including: • **Vectorization** • **Loop unrolling**
`-O3`	**Performs O2 optimizations and enables more aggressive loop transformations such as:** • **Loop fusion** • **Block unroll-and-jam** • **Collapsing IF statements** This option is recommended for applications that have loops that heavily use floating-point calculations and process large data sets. However, it might incur in slower code, numerical stability issues, and compilation time increase.

Vectorization: target architecture options

Option	Description
`-mmic`	Builds an application that runs natively on Intel® MIC Architecture.
`-xfeature` `-xHost`	Tells the compiler which processor features it may target, referring to which instruction sets and optimizations it may generate (not available for Intel® Xeon Phi™ architecture). Values for *feature* are: • `COMMON-AVX512` (includes AVX512 FI and CDI instructions) • **`MIC-AVX512` (includes AVX512 FI, CDI, PFI, and ERI instructions)** • **`CORE-AVX512` (includes AVX512 FI, CDI, BWI, DQI, and VLE instructions)** • **`CORE-AVX2`** • `CORE-AVX-I` (including RDRND instruction) • `AVX` • `SSE4.2, SSE4.1` • `ATOM_SSE4.2, ATOM_SSSE3` (including MOVBE instruction) • `SSSE3, SSE3, SSE2` **When using `-xHost`, the compiler will generate instructions for the highest instruction set available on the compilation host processor.**
`-axfeature`	Tells the compiler to generate multiple, feature-specific auto-dispatch code paths for Intel® processors if there is a performance benefit. Values for *feature* are the same described for `-xfeature` option. Multiple features/paths possible, e.g.: `-axSSE2,AVX`. It also generates a baseline code path for the default case.

Auto vectorization: not all loops will vectorize

- Data dependencies between iterations
  - Proven Read-after-Write data (i.e., loop carried) dependencies
  - Assumed data dependencies
    - ❑ Aggressive optimizations

RaW dependency

```
for (int i = 0; i < N; i++)
    a[i] = a[i-1] + b[i];
```

- Vectorization won’t be efficient
  - Compiler estimates how better the vectorized version will be
  - Affected by data alignment, data layout, etc.

Inefficient vectorization

```
for (int i = 0; i < N; i++)
    a[c[i]] = b[d[i]];
```

- Unsupported loop structure
  - While-loop, for-loop with unknown number of iterations
  - Complex loops, unsupported data types, etc.
  - (Some) function calls within loop bodies

Function call within loop body

```
for (int i = 0; i < N; i++)
    a[i] = foo(b[i]);
```

Validating vectorization

- Generate compiler report about optimizations

`-qopt-report[=n]` Generate report (level [1..6], default 2)

```
LOOP BEGIN at gas_dyn2.f90(193,11) inlined into gas_dyn2.f90(4326,31)
   remark #15300: LOOP WAS VECTORIZED
   remark #15448: unmasked aligned unit stride loads: 1
   remark #15450: unmasked unaligned unit stride loads: 1
   remark #15475: --- begin vector loop cost summary ---
   remark #15476: scalar loop cost: 53
   remark #15477: vector loop cost: 14.870
   remark #15478: estimated potential speedup: 2.520
   remark #15479: lightweight vector operations: 19
   remark #15481: heavy-overhead vector operations: 1
   remark #15488: --- end vector loop cost summary ---
   remark #25456: Number of Array Refs Scalar Replaced In Loop: 1
   remark #25015: Estimate of max trip count of loop=4
LOOP END
```

Vectorized loop

```
LOOP BEGIN at gas_dyn2.f90(2346,15)
   remark #15344: loop was not vectorized: vector dependence prevents vectorization
   remark #15346: vector dependence: assumed OUTPUT dependence between IOLD line 376 and IOLD line 354
   remark #25015: Estimate of max trip count of loop=3000001
LOOP END
```

Non-vectorized loop

Agenda

- Vector Processing Units;
- Profiling;
- Optimizing Memory Access;
- Auto Vectorization;
- **Guided Vectorization.**

Intel® compiler directives for vectorization

Directive	Clause	Description
`ivdep`		Instructs the compiler to ignore assumed vector dependencies.
`vector`	`always`	Force vectorization even when it might be not efficient.
	`[un]aligned`	Use [un]aligned data movement instructions for all array vector references.
	`[non]temporal(`*`var1`*`[,…])`	Do or do not generate non-temporal (streaming) stores for the given array variables. On Intel® MIC architecture, generates a cache-line-evict instruction when the store is known to be aligned.
	`[no]vecreminder`	Do (not) vectorize the remainder loop when the main loop is vectorized.
	`[no]mask_readwrite`	Enables/disables memory speculation causing the generation of [non-]masked loads and stores within conditions.
`simd`	`vectorlength(`*`n1`*`[,…])` `vectorlengthfor(`*`dtype`*`)`	Assume safe vectorization for the given vector length values or data type.

	`private(`*`var1`*`[,…])`
	`firstprivate(`*`var1`*`[,…])`
	`lastprivate(`*`var1`*`[,…])`
		Which variables are private to each iteration; *firstprivate*, initial value is broadcasted to all private instances; *lastprivate*, last value is copied out from the last instance.

	`linear(`*`var1`*`:`*`step1`*`[,…])`	Letting know the compiler that *var1* is incremented by *step1* on every iteration of the original loop.
	`reduction(`*`oper`*`:`*`var1`*`[,…])`	Which variables are reduction variables with a given operator.
	`[no]assert`	Warning or error when vectorization fails.
	`[no]vecremainder`	Do (not) vectorize the remainder loop when the mail loop is vectorized.

Guided vectorization: disambiguation hints

- Assume function arguments won't be aliased
  - C/C++: Compile with `-fargument-noalias`
- C99 "restrict" keyword for pointers
  - Compile with `-restrict` otherwise

```
void v_add(float *restrict c,
           float *restrict a,
           float *restrict b)
{
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

```
void v_add(float *c, float *a, float *b)
{
for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

Guided vectorization:

- `#pragma simd` or `#pragma ivdep`
  - Force loop vectorization ignoring **all** dependencies
    - ❑ Additional clauses for specify reductions, etc.

```
void v_add(float *c, float *a, float *b)
{
#pragma simd
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

SIMD loop

```
__declspec(vector)
void v_add(float c, float a, float b)
{
    c = a + b;
}
    …
for (int i = 0; i < N; i++)
    v_add(C[i], A[i], B[i]);
```

SIMD function

Matrix Multiplication - Serial

```
void multiply(int msize, int tidx, int numt, TYPE a[][NUM], TYPE
b[][NUM], TYPE c[][NUM], TYPE t[][NUM])
{

int i,j,k;
    for(i=0; i<msize; i++) {
        for(k=0; k<msize; k++) {
            for(j=0; j<msize; j++) {
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }
    }
}
```

Matrix Multiplication

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?
[loop in multiply3 at multiply.c:228]		2 Assume ...	0.170s	0.170s	Scalar	vector dependence prevents vectorization
[loop in __libc_csu_init]			0.000s	0.000s	Scalar	
[loop in _INTERNAL_16_offload_host_cpp_ad92 ...			0.000s	0.000s	Scalar	
[loop in func@0x5b810]		2 Data typ ..	0.000s	0.000s	Scalar	
[loop in main at matrix.c:144]		2 Data typ ..	0.000s	0.000s	Scalar	inner loop was already vectorized
[loop in multiply3 at multiply.c:227]		2 Assume ...	0.000s	0.170s	Scalar	vector dependence prevents vectorization
[loop in multiply3 at multiply.c:226]		2 Assume ...	0.000s	0.170s	Scalar	vector dependence prevents vectorization
[loop in main at matrix.c:144]		1 Data typ ..	0.000s	0.000s	Vectorized (B ...	

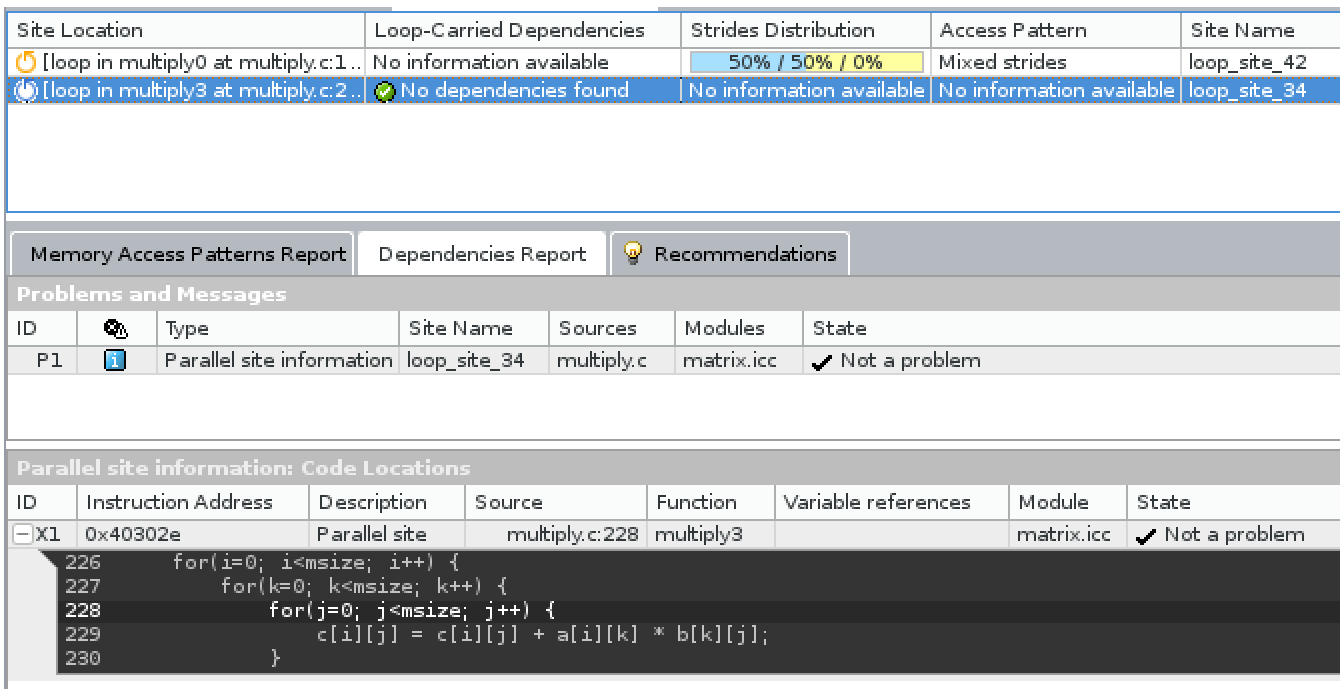
Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: multiply.c:228 multiply3

```
Line  Source
218   void multiply3(int msize, int tidx, int numt, TYPE a[][NUM], TYPE b[][NUM], TYPE c[][NUM], TYPE t[][NUM])
219   {
220
221   //#pragma omp target device(0) map(a[0:NUM][0:NUM]) \
222     //map(b[0:NUM][0:NUM]) map(c[0:NUM][0:NUM])
223     //{
224       int i,j,k;
225   //    #pragma omp parallel for collapse (2) //num_threads(60)
226       for(i=0; i<msize; i++) {
          [loop in multiply3 at multiply.c:226]
            Scalar loop. Not vectorized: vector dependence prevents vectorization
            No loop transformations applied
227           for(k=0; k<msize; k++) {
          [loop in multiply3 at multiply.c:227]
            Scalar loop. Not vectorized: vector dependence prevents vectorization
            Remainder loop
228               for(j=0; j<msize; j++) {
          [loop in multiply3 at multiply.c:228]
            Scalar loop. Not vectorized: vector dependence prevents vectorization
            Loop was unrolled by 2
229                   c[i][j] = c[i][j] + a[i][k] * b[k][j];
230               }
231           }
232       }
233     //}
234   }
235
```

Matrix Multiplication

- Check dependency analysis shows that it is safe to enforce the vectorization of this loop

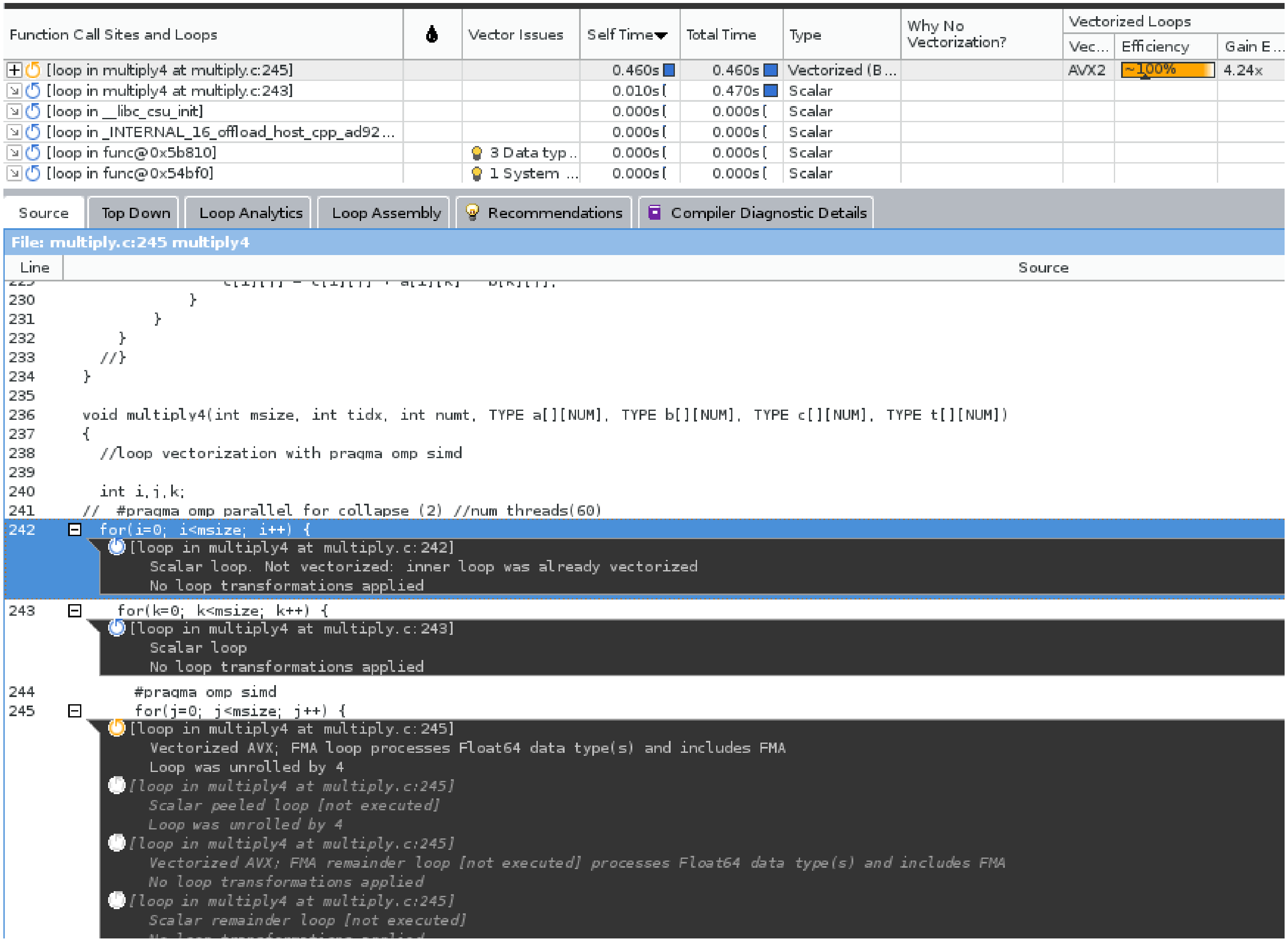


Matrix Multiplication - vectorized

```
void multiply(int msize, int tidx, int numt, TYPE a[][NUM], TYPE
b[][NUM], TYPE c[][NUM], TYPE t[][NUM])
{

int i,j,k;
    for(i=0; i<msize; i++) {
        for(k=0; k<msize; k++) {
            #pragma simd
            for(j=0; j<msize; j++) {
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }
    }
}
```

Matrix Multiplication



Example

- Particle Binning Problem**[1]**

- Optimizations:
  - Automatic Vectorization
  - Data Alignment

[1] http://colfaxresearch.com/optimization-techniques-for-the-intel-mic-architecture-part-2-of-3-strip-mining-for-vectorization/

Particle Binning - Serial

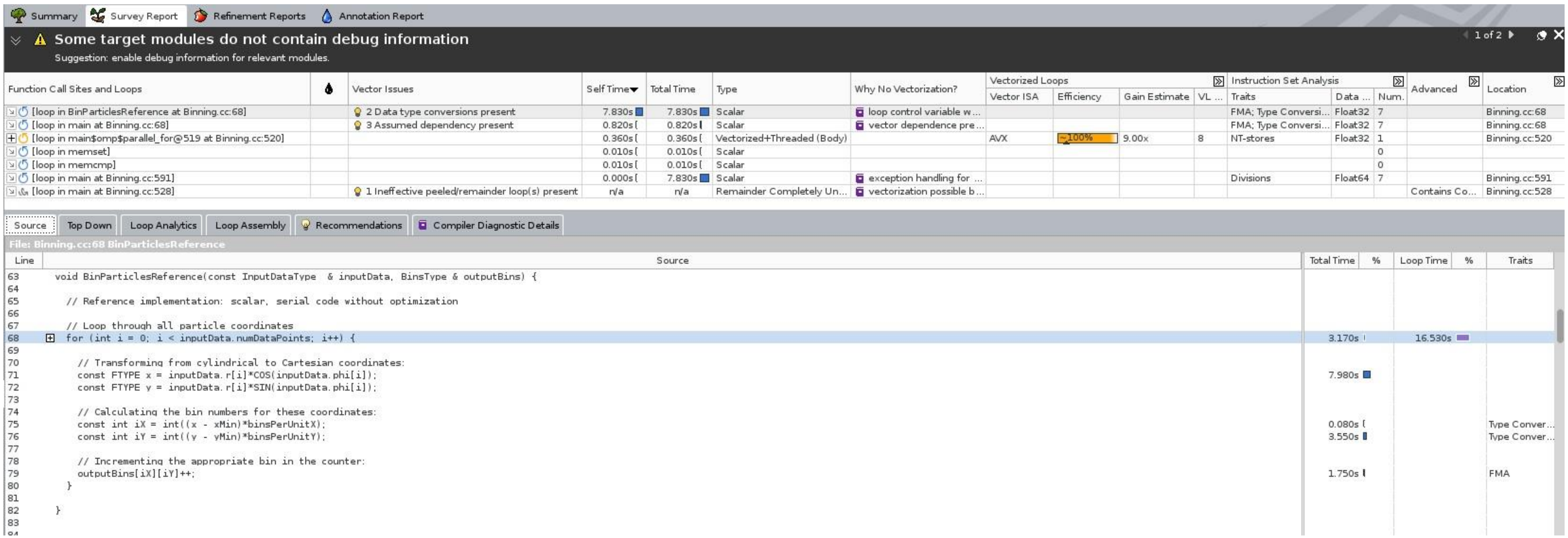
```
for (int i = 0; i < inputData.numDataPoints; i++) {

    // Transforming from cylindrical to Cartesian coordinates:
    const FTYPE x = inputData.r[i]*COS(inputData.phi[i]);
    const FTYPE y = inputData.r[i]*SIN(inputData.phi[i]);

    // Calculating the bin numbers for these coordinates:
    const int iX = int((x - xMin)*binsPerUnitX);
    const int iY = int((y - yMin)*binsPerUnitY);

}
```

Particle Binning - Serial



Particle Binning - Vectorized

```
for (int ii = 0; ii < inputData.numDataPoints; ii += STRIP_WIDTH) {

    int iX[STRIP_WIDTH];
    int iY[STRIP_WIDTH];

    const FTYPE* r   = &(inputData.r[ii]);
    const FTYPE* phi = &(inputData.phi[ii]);

    // Vector loop

    for (int c = 0; c < STRIP_WIDTH; c++) {
        // Transforming from cylindrical to Cartesian coordinates:
        const FTYPE x = r[c]*COS(phi[c]);
        const FTYPE y = r[c]*SIN(phi[c]);

        // Calculating the bin numbers for these coordinates:
        iX[c] = int((x - xMin)*binsPerUnitX);
        iY[c] = int((y - yMin)*binsPerUnitY);
    }

}
```

Particle Binning - Vectorized

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops: Vector ISA	Efficiency	Gain Estimate	VL ...	Instruction Set Analysis: Traits	Data ...	Num.	Advanced	Location
[loop in main at Binning.cc:68]		3 Assumed dependency present	0.880s	0.880s	Scalar	vector dependence pre...					FMA; Type Conversi...	Float32	7		Binning.cc:68
[loop in BinParticles_3 at Binning.cc:173]		1 Potential underutilization of FMA instructions	0.850s	1.420s	Scalar	inner loop was already ...							9		Binning.cc:173
[loop in BinParticles_3 at Binning.cc:182]		1 Data type conversions present	0.570s	0.570s	Vectorized (Body)		AVX2	~100%	17.64x	8	FMA; Type Conversi...	Float3...	9		Binning.cc:182
[loop in mainompparallel_for@519 at Binning.cc:520]			0.359s	0.359s	Vectorized+Threaded (Body)		AVX	~100%	9.00x	8	NT-stores	Float32	1		Binning.cc:520
[loop in operator new]			0.010s	0.010s	Scalar								0		
[loop in memcmp]			0.010s	0.010s	Scalar								0		
[loop in main at Binning.cc:591]			0.000s	1.420s	Scalar	exception handling for ...					Divisions	Float64	8		Binning.cc:591
[loop in main at Binning.cc:528]		1 Ineffective peeled/remainder loop(s) present	n/a	n/a	Remainder Completely Un...	vectorization possible b...								Contains Co...	Binning.cc:528

Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: Binning.cc:68 main

Line	Source	Total Time	%	Loop Time	%	Traits
169	`threadPrivateBins[i][j] = 0;`					
170						
171	`// Loop through all bunches of particles`					
172	`//#pragma omp for`					
173	`for (int ii = 0; ii < inputData.numDataPoints; ii += STRIP WIDTH) {`	0.050s		4.250s		
174						
175	`int iX[STRIP WIDTH];`					
176	`int iY[STRIP WIDTH];`					
177						
178	`const FTYPE* r   = &(inputData.r[ii]);`					
179	`const FTYPE* phi = &(inputData.phi[ii]);`	0.010s				
180						
181	`// Vector loop`					
182	`for (int c = 0; c < STRIP_WIDTH; c++) {` [loop in BinParticles_3 at Binning.cc:182] Vectorized AVX; FMA loop processes Float32; Int32 data type(s) and includes FMA; Type Conversions Loop stmts were reordered	0.060s		3.400s		
183	`// Transforming from cylindrical to Cartesian coordinates:`					
184	`const FTYPE x = r[c]*COS(phi[c]);`					
185	`const FTYPE y = r[c]*SIN(phi[c]);`	3.040s				

Particle Binning - Data Alignment

```
for (int ii = 0; ii < inputData.numDataPoints; ii += STRIP_WIDTH) {

    int iX[STRIP_WIDTH] __attribute__((aligned(64)));
    int iY[STRIP_WIDTH] __attribute__((aligned(64)));

    const FTYPE* r   = &(inputData.r[ii]);
    const FTYPE* phi = &(inputData.phi[ii]);

    // Vector loop
#pragma vector aligned
    for (int c = 0; c < STRIP_WIDTH; c++) {
        // Transforming from cylindrical to Cartesian coordinates:
        const FTYPE x = r[c]*COS(phi[c]);
        const FTYPE y = r[c]*SIN(phi[c]);

        // Calculating the bin numbers for these coordinates:
        iX[c] = int((x - xMin)*binsPerUnitX);
        iY[c] = int((y - yMin)*binsPerUnitY);
    }

}
```

Particle Binning - Data Alignment

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops: Vector ISA	Efficiency	Gain Estimate	VL ...	Instruction Set Analysis: Traits	Data ...	Num.	Advanced	Location
[loop in BinParticles_4 at Binning.cc:226]		1 Potential underutilization of FMA instructions	0.940s	1.470s	Scalar	inner loop was already ...							9		Binning.cc:226
[loop in main at Binning.cc:68]		3 Assumed dependency present	0.900s	0.900s	Scalar	vector dependence pre ...					FMA; Type Conversi...	Float32	7		Binning.cc:68
[loop in BinParticles_4 at Binning.cc:236]		1 Data type conversions present	0.530s	0.530s	Vectorized (Body)		AVX2	~100%	18.50x	8	FMA; Type Conversi...	Float3...	9		Binning.cc:236
[loop in mainompparallel_for@519 at Binning.cc:520]			0.370s	0.370s	Vectorized+Threaded (Body)		AVX	~100%	9.00x	8	NT-stores	Float32	1		Binning.cc:520
[loop in memcmp]			0.010s	0.010s	Scalar								0		
[loop in main at Binning.cc:591]			0.000s	1.470s	Scalar	exception handling for ...					Divisions	Float64	8		Binning.cc:591
[loop in main at Binning.cc:528]		1 Ineffective peeled/remainder loop(s) present	n/a	n/a	Remainder Completely Un...	vectorization possible b ...								Contains Co...	Binning.cc:528

Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: Binning.cc:226 BinParticles_4

Line	Source	Total Time	%	Loop Time	%	Traits
232	`const FTYPE* phi = &(inputData.phi[ii]);`	0.010s				
233						
234	`// Vector loop`					
235	`#pragma vector aligned`					
236	`for (int c = 0; c < STRIP_WIDTH; c++) {`			3.250s		
	[loop in BinParticles_4 at Binning.cc:236] Vectorized AVX; FMA loop processes Float32; Int32 data type(s) and includes FMA; Type Conversions No loop transformations applied					
237	`// Transforming from cylindrical to Cartesian coordinates:`					
238	`const FTYPE x = r[c]*COS(phi[c]);`	0.090s				
239	`const FTYPE y = r[c]*SIN(phi[c]);`	2.740s				
240						
241	`// Calculating the bin numbers for these coordinates:`					
242	`iX[c] = int((x - xMin)*binsPerUnitX);`	0.110s				FMA; Type C ...
243	`iY[c] = int((y - yMin)*binsPerUnitY);`	0.310s				FMA; Type C ...
244	`}`					
245						
246	`// Scalar loop`					
247	`for (int c = 0; c < STRIP_WIDTH; c++)`					
248	`threadPrivateBins[iX[c]][iY[c]]++;`	0.890s				

Diffusion - Serial

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?
[loop in diffusion_baseline at diffusion_base.c:103]		1 Potential ..	0.000s	10.530s	Scalar	outer loop was not auto-vectorized: consider using SIMD ...
[loop in diffusion_baseline at diffusion_base.c:104]		1 Potential ..	0.010s	10.450s	Scalar	outer loop was not auto-vectorized: consider using SIMD ...
[loop in diffusion_baseline at diffusion_base.c:105]		2 Assume ...	0.130s	10.370s	Scalar	vector dependence: assumed dependence between lines
[loop in diffusion_baseline at diffusion_base.c:105]		2 Assume ...	0.000s	0.080s	Scalar Versions	1 vector dependence: assumed dependence between lines
[loop in diffusion_baseline at diffusion_base.c:107]		2 Assume ...	10.240s	10.240s	Scalar	vector dependence: assumed dependence between lines
[loop in diffusion_baseline at diffusion_base.c:107]		2 Assume ...	0.070s	0.070s	Scalar Versions	1 vector dependence: assumed dependence between lines
[loop in diffusion_baseline at diffusion_base.c:107]		2 Assume ...	0.060s	0.060s	Scalar	vector dependence: assumed dependence between lines

Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: diffusion_base.c:107 diffusion_baseline

Line	Source

```
97      void
98      diffusion_baseline(REAL *f1, REAL *f2, int nx, int ny, int nz,
99                         REAL ce, REAL cw, REAL cn, REAL cs, REAL ct,
100                        REAL cb, REAL cc, REAL dt,
101                        int count) {
102       int i;
103       for (i = 0; i < count; ++i) {
104         for (int z = 0; z < nz; z++) {
105           for (int y = 0; y < ny; y++) {
106             //#pragma simd
107             for (int x = 0; x < nx; x++) {
          [loop in diffusion_baseline at diffusion_base.c:107]
             Scalar loop. Vector dependence: assumed dependence between lines
             No loop transformations applied
          [loop in diffusion_baseline at diffusion_base.c:107]
             Scalar loop. Vector dependence: assumed dependence between lines
             No loop transformations applied
          [loop in diffusion_baseline at diffusion_base.c:107]
             Scalar loop. Vector dependence: assumed dependence between lines
             No loop transformations applied
          [loop in diffusion_baseline at diffusion_base.c:107]
             Scalar loop. Vector dependence: assumed dependence between lines
             No loop transformations applied
          [loop in diffusion_baseline at diffusion_base.c:107]
             Scalar loop. Vector dependence: assumed dependence between lines
             No loop transformations applied
```

Diffusion - Vectorized

Potential inefficient memory access;

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops		
							Vec...	Efficiency	Gai...
[loop in diffusion_openmpv at diffusion_ompvect.c:106]		1 Potential..	0.640s	5.050s	Scalar	inner loop was already vectorized			
[loop in diffusion_openmpv at diffusion_ompvect.c:106]		1 Potential..	0.000s	0.060s	Scalar Versions	1 inner loop was already vectorized			
[loop in diffusion_openmpv at diffusion_ompvect.c:108]			4.410s	4.410s	Vectorized (B...		AVX2	~62%	4.93x
[loop in diffusion_openmpv at diffusion_ompvect.c:108]		1 Possible i..	0.060s	0.060s	Vectorized Ve...		AVX2	~50%	4.00x
[loop in diffusion_openmpv at diffusion_ompvect.c:108]		1 Possible i..	0.030s	0.030s	Vectorized (B...		AVX2	~50%	4.00x
[loop in diffusion_openmpv at diffusion_ompvect.c:108]		1 Possible i..	0.030s	0.030s	Vectorized (B...		AVX2	~50%	4.00x
[loop in init at diffusion_ompvect.c:71]		1 Data typ..	0.000s	0.060s	Scalar	inner loop was already vectorized			

Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: diffusion_ompvect.c:108 diffusion_openmpv

```
Line  Source
97    void
98    diffusion_openmpv(REAL *restrict f1, REAL *restrict f2, int nx, int ny, int nz,
99                      REAL ce, REAL cw, REAL cn, REAL cs, REAL ct,
100                     REAL cb, REAL cc, REAL dt, int count) {
101       REAL *f1_t = f1;
102       REAL *f2_t = f2;
103
104       for (int i = 0; i < count; ++i) {
105         for (int z = 0; z < nz; z++) {
106           for (int y = 0; y < ny; y++) {
107         #pragma simd
108             for (int x = 0; x < nx; x++) {
        [loop in diffusion_openmpv at diffusion_ompvect.c:108]
          Vectorized AVX; FMA loop processes Float32 data type(s) and includes FMA
          No loop transformations applied
        [loop in diffusion_openmpv at diffusion_ompvect.c:108]
          Vectorized AVX; AVX2; AVX2GATHER; FMA loop processes Float32; Int32 data type(s) and includes FMA; Gathers
          No loop transformations applied
        [loop in diffusion_openmpv at diffusion_ompvect.c:108]
          Vectorized AVX; AVX2; AVX2GATHER; FMA loop processes Float32; Int32 data type(s) and includes FMA; Gathers
          No loop transformations applied
        [loop in diffusion_openmpv at diffusion_ompvect.c:108]
          Vectorized AVX; AVX2; AVX2GATHER; FMA loop processes Float32; Int32 data type(s) and includes FMA; Gathers
          No loop transformations applied
        [loop in diffusion_openmpv at diffusion_ompvect.c:108]
          Vectorized AVX; AVX2; AVX2GATHER; FMA loop processes Float32; Int32 data type(s) and includes FMA; Gathers
          No loop transformations applied
```

Diffusion - alignment

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops: Vec...	Efficiency	Gai...
[loop in accuracy at diffusion_peel.c:91]			0.010s	0.010s	Vectorized (B...		AVX2	~100%	12. ...
[loop in diffusion_peel at diffusion_peel.c:106]		1 Potential underutilization of FMA instructions	0.000s	5.240s	Scalar	inner loop was already vectorized			
[loop in diffusion_peel at diffusion_peel.c:108]		1 Potential underutilization of FMA instructions	0.000s	5.240s	Scalar	inner loop was already vectorized			
[loop in diffusion_peel at diffusion_peel.c:109]		1 Potential underutilization of FMA instructions	0.580s	5.240s	Scalar	inner loop was already vectorized			
[loop in diffusion_peel at diffusion_peel.c:120]			**4.660s**	**4.660s**	**Vectorized ...**		**AVX.**	~99%	**7.8..**
[loop in init at diffusion_peel.c:71]		1 Data type conversions present	0.000s	0.070s	Scalar	inner loop was already vectorized			

Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: diffusion_peel.c:120 diffusion_peel

```
Line  Source                                                                Total Time  %  Loop Time  %
96
97    void
98    diffusion_peel(REAL *restrict f1, REAL *restrict f2, int nx, int ny, int nz,
99                   REAL ce, REAL cw, REAL cn, REAL cs, REAL ct,
100                  REAL cb, REAL cc, REAL dt, int count) {
101   //#pragma omp parallel
102    // {
103       REAL *f1_t = f1;
104       REAL *f2_t = f2;
105
106       for (int i = 0; i < count; ++i) {                                                     5.240s
107   //#pragma omp for collapse(2)
108         for (int z = 0; z < nz; z++) {                                                      5.240s
109           for (int y = 0; y < ny; y++) {                                                    5.240s
110         int x, c, n, s, b, t;
111             x = 0;
112             c =  x + y * NXP + z * NXP * ny;
113             n = (y == 0)    ? c : c - NXP;
114             s = (y == ny-1) ? c : c + NXP;                          0.020s
115             b = (z == 0)    ? c : c - NXP * ny;
116             t = (z == nz-1) ? c : c + NXP * ny;
117             f2_t[c] = cc * f1_t[c] + cw * f1_t[c] + ce * f1_t[c+1]  0.010s
118                 + cs * f1_t[s] + cn * f1_t[n] + cb * f1_t[b] + ct * f1_t[t];  0.030s
119   #pragma simd
120             for (x = 1; x < nx-1; x++) {                            0.140s         4.660s
              [loop in diffusion_peel at diffusion_peel.c:120]
                Vectorized AVX; FMA loop processes Float32 data type(s) and includes FMA
                No loop transformations applied
121               ++c;
122               ++n;                                                  0.010s
123               ++s;
124               ++b;
125               ++t;
126               f2_t[c] = cc * f1_t[c] + cw * f1_t[c-1] + ce * f1_t[c+1]   0.760s
127                   + cs * f1_t[s] + cn * f1_t[n] + cb * f1_t[b] + ct * f1_t[t];  4.240s
128             }
129             ++c;
130             ++n;
131             ++s;
132             ++b;
133             ++t;
134             f2_t[c] = cc * f1_t[c] + cw * f1_t[c-1] + ce * f1_t[c]
135                 + cs * f1_t[s] + cn * f1_t[n] + cb * f1_t[b] + ct * f1_t[t];  0.030s
136           }
137         }
138         REAL *t = f1_t;
139         f1_t = f2_t;
```

Interpolation

```
__declspec(vector)
int FindPosition(double x) {
    return (int)(log(exp(x*steps)));
}

__declspec(vector)
double Interpolate(double x, const point* vals)
{
    int ind = FindPosition(x);
    ...

    return res;
}
```

```
int main ( int argc , char argv [] )
{
    . . .
  for ( i=0; i <ARRAY_SIZE;++ i ) {
      dst[i] = Interpolate( src[i], vals ) ;
  }
    . . .
}
```

George M. Raskulinec, Evgeny Fiksman “Chapter 22 - **SIMD functions via OpenMP**”, In High Performance Parallelism Pearls, edited by James Reinders and Jim Jeffers, Morgan Kaufmann, Boston, 2015, Pages 171-190, ISBN 9780128038192

Vectorization report - Interpolate

```
Begin optimization report for: Interpolate.._simdsimd3__H2n_v1_s1.P(double, const point *)

    Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED   [ main.c(74,48) ]
===========================================================================

Begin optimization report for: Interpolate.._simdsimd3__H2m_v1_s1.P(double, const point *)

    Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED   [ main.c(74,48) ]
===========================================================================

Begin optimization report for: Interpolate.._simdsimd3__L4n_v1_s1.V(double, const point *)

    Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED   [ main.c(74,48) ]
remark #15415: vectorization support: gather was generated for the variable pnt:  indirect access, 64bit indexed   [ main.c(78,26) ]
remark #15415: vectorization support: gather was generated for the variable pnt:  indirect access, 64bit indexed   [ main.c(78,36) ]
===========================================================================

Begin optimization report for: Interpolate.._simdsimd3__L4m_v1_s1.V(double, const point *)

    Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED   [ main.c(74,48) ]
remark #15415: vectorization support: gather was generated for the variable pnt: masked, indirect access, 64bit indexed   [ main.c(78,26) ]
remark #15415: vectorization support: gather was generated for the variable pnt: masked, indirect access, 64bit indexed   [ main.c(78,36) ]
```

Vectorization report - FindPosition

egin optimization report for: FindPosition.._simdsimd3__H2n_v1.P(double)

Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED [main.c(70,28)]
===

Begin optimization report for: FindPosition.._simdsimd3__H2m_v1.P(double)

Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED [main.c(70,28)]
===

Begin optimization report for: FindPosition.._simdsimd3__L4n_v1.V(double)

Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED [main.c(70,28)]
===

Begin optimization report for: FindPosition.._simdsimd3__L4m_v1.V(double)

Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED [main.c(70,28)]
===

Lattice Boltzmann

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops			
							Vec...	Efficiency	Gai...	VL ...
[loop in fCalcInteraction_ShanChen_Boundary at lbpFORCE.c...		2 Assume...	0.130s	0.210s	Scalar	vector depende...				
[loop in fCalcInteraction_ShanChen_Boundary at lbpFORCE.c...		2 Assume...	0.080s	0.080s	Scalar	vector depende...				
[loop in fCalcPotential_ShanChen at lbpFORCE.cpp:36]			0.000s	5.742s	Scalar	loop control vari...				
[loop in fCalcPotential_ShanChen at lbpFORCE.cpp:49]		1 Potential..	0.000s	5.742s	Threaded (OpenMP)	inner loop was al...				
[loop in fCollisionBGKompparallel@338 at lbpBGK.cpp:351]		1 Ineffecti...	0.160s	0.160s	Vectorized (Remainder)		AVX2	~67%	2.69x	4
[loop in fGetEquilibriumF at lbpSUB.cpp:615]		**2 Data t...**	**3.910s**	**3.910s**	**Vectorized (Body; ...**		**AVX.**	~91%	**3.6..**	**4**
[loop in fGetEquilibriumF at lbpSUB.cpp:615]		1 Data typ..	3.120s	3.120s	Vectorized (Body)		AVX2			4
[loop in fGetEquilibriumF at lbpSUB.cpp:615]		1 Data typ..	0.790s	0.790s	Remainder					

Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: lbpSUB.cpp:615 fGetEquilibriumF

```
Line  Source
600
601
602
603
604   int fGetEquilibriumF(double *feq, double *v, double rho)
605   {
606
607     // calculate equilibrium distribution function for compressible fluid:
608     // only suitable for square lattices, not suitable for incompressible fluid
609
610     double modv = v[0]*v[0] + v[1]*v[1] + v[2]*v[2];
611     double uv;
612
613       //#pragma loop count(20)
614       //for(int i=0; i<lbsy.nq +1; i++)
615       for(int i=0; i<lbsy.nq; i++)
        [loop in fGetEquilibriumF at lbpSUB.cpp:615]
          Scalar loop. Not vectorized: vector dependence prevents vectorization
          No loop transformations applied
        [loop in fGetEquilibriumF at lbpSUB.cpp:615]
          Vectorized AVX; FMA loop processes Float64; Int32; UInt32 data type(s) and includes FMA; Type Conversions; Unpacks
          No loop transformations applied
        [loop in fGetEquilibriumF at lbpSUB.cpp:615]
          Scalar remainder loop
          No loop transformations applied
```

Lattice Boltzmann

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops: Vec...	Efficiency	Gai...	VL ...
[loop in fCalcPotential_ShanChen at lbpFORCE.cpp:49]		1 Potential...	0.000s	5.871s	Threaded (OpenMP)	inner loop was al...				
[loop in fCollisionBGKompparallel@338 at lbpBGK.cpp:351]		1 Ineffecti...	0.290s	0.290s	Vectorized (Remainder)		AVX2	~67%	2.69x	4
[loop in fGetEquilibriumF at lbpSUB.cpp:614]		**1 Data t...**	**4.000s**	**4.000s**	**Vectorized (Body)**		**AVX.**	**~88%**	**3.5..**	**4**
[loop in fGetEquilibriumF at lbpSUB.cpp:614]		1 Data typ...	4.000s	4.000s	Vectorized (Body)		AVX2			4
[loop in fGetEquilibriumF at lbpSUB.cpp:614]		3 Assume...	0.030s	0.030s	Scalar	vector depende...				
[loop in fGetEquilibriumF at lbpSUB.cpp:614]		3 Assume...	0.010s	0.010s	Scalar	vector depende...				
[loop in fGetFracSite at lbpGET.cpp:121]		2 Ineffecti...	0.210s	0.210s	Vectorized Versions		AVX...	~31%	2.50x	4; 8
[loop in fGetOneDirecSpeedSite at lbpGET.cpp:241]			0.030s	0.140s	Scalar					

Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

File: lbpSUB.cpp:614 fGetEquilibriumF

```
Line  Source
603
604   int fGetEquilibriumF(double *feq, double *v, double rho)
605   {
606
607     // calculate equilibrium distribution function for compressible fluid:
608     // only suitable for square lattices, not suitable for incompressible fluid
609
610     double modv = v[0]*v[0] + v[1]*v[1] + v[2]*v[2];
611     double uv;
612
613       #pragma loop count(20)
614       for(int i=0; i<lbsy.nq +1; i++)
        [loop in fGetEquilibriumF at lbpSUB.cpp:614]
          Scalar loop. Not vectorized: vector dependence prevents vectorization
          No loop transformations applied
        [loop in fGetEquilibriumF at lbpSUB.cpp:614]
          Scalar loop. Not vectorized: vector dependence prevents vectorization
          No loop transformations applied
        [loop in fGetEquilibriumF at lbpSUB.cpp:614]
          Vectorized AVX; FMA loop processes Float64; Int32; UInt32 data type(s) and includes FMA; Type Conversions; Unpacks
          No loop transformations applied
        [loop in fGetEquilibriumF at lbpSUB.cpp:614]
          Scalar remainder loop [not executed]
          No loop transformations applied
```

Questions?

Silvio Stanzani , Raphael Cóbe , Rogério Iope
Núcleo de Computação Científica – UNESP

silvio@ncc.unesp.br
rmcobe@ncc.unesp.br
rogerio@ncc.unesp.br