Hands-on “Vectorization”

- Automatic Vectorization
  - Vectorization Report
- Guided Vectorization
  - pragma simd
  - pragma ivdep
  - restrict
- Intel Advisor
  - Create Advisor Project
  - Collect Survey Data
  - Change xhost
- Matrix Multiplication
  - New advisor Project
  - Collect Survey Data
  - Check Trip Count
  - Check Dependencies
  - Memory Access Patterns

1) Compile the example with vec-report6 and O3

```
icc VectorizationHandson.c -o VectorizationHandson -vec-report6 –O3
```

2) Open the vectorization report (VectorizationHandson.optrpt)

Note that the loop on function main was automatically vectorized;

```
remark #15300: LOOP WAS VECTORIZED
```

Note that the loop on function add_floats was not automatically vectorized;

```
remark #15344: loop was not vectorized: vector dependence prevents vectorization
```

2) Include “#pragma ivdep directive” in top of loop “for (i=0; i<n; i++)“ on function add_floats

```
#pragma ivdep
for (i=0; i<n; i++){
```

3) Recompile the example with vec-report

```
icc VectorizationHandson.c -o VectorizationHandson -vec-report6 –O3
```

Note that the Outer loop can be vectorized now.

```
remark #15301: OUTER LOOP WAS VECTORIZED
```

2) Include pragma simd directive in the loop “for (j=n; j>0; j--){“ on function add_floats

```
#pragma simd
for (j=n; j>0; j--){
```

3) Recompile the example with vec-report

```
icc VectorizationHandson.c -o VectorizationHandson -vec-report6 –O3
```

Note that compiler vectorized the inner loop instead of outer loop

Outer loop:

```
remark #15542: loop was not vectorized: inner loop was already vectorized
```

Inner loop:

```
remark #15301: SIMD LOOP WAS VECTORIZED
```

2) Include keyword restrict to avoid in all arguments of function quad

```
void quad(int length, double * restrict a, double * restrict b, double * restrict c, double * 
restrict  x1, double * restrict  x2)
```

3) Recompile the example with vec-report and restrict

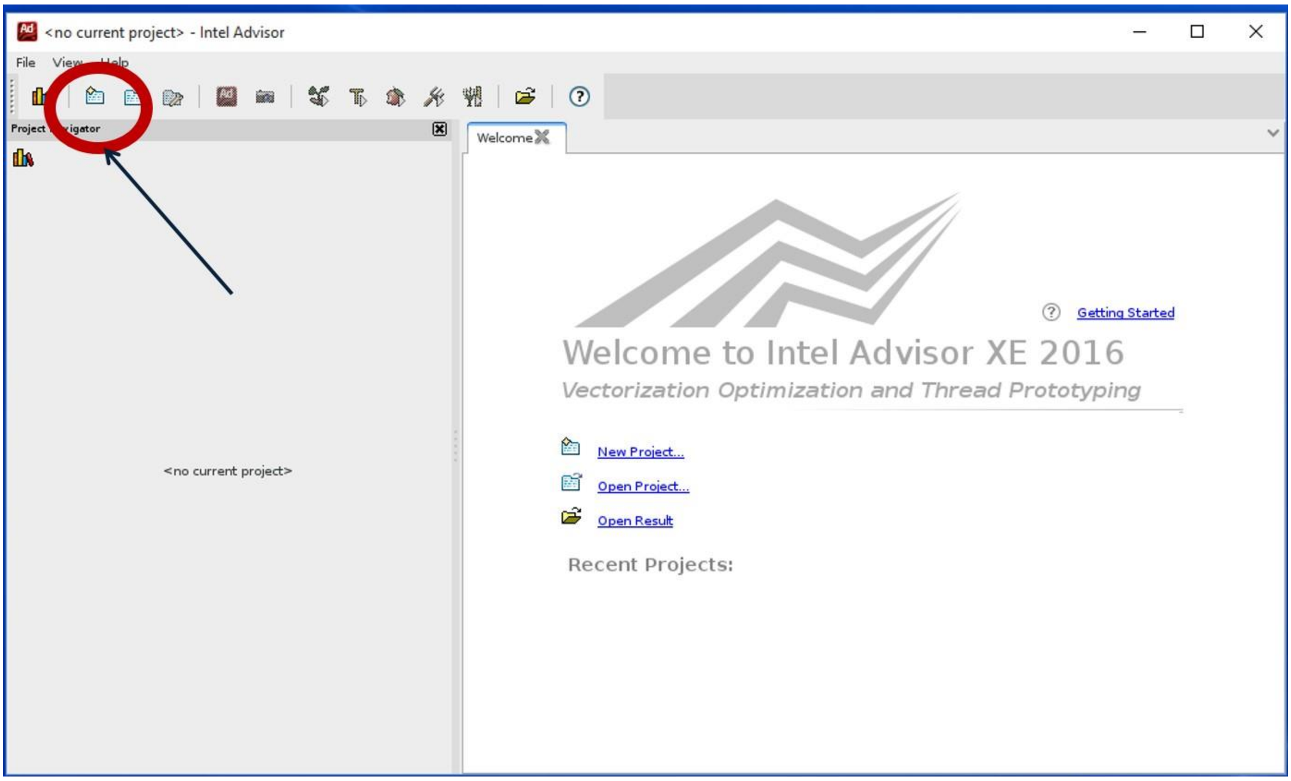
```
icc VectorizationHandson.c -o VectorizationHandson -vec-report6 –O3 -restrict
```

Note that the loop can be vectorized now.

```
remark #15300: LOOP WAS VECTORIZED
```

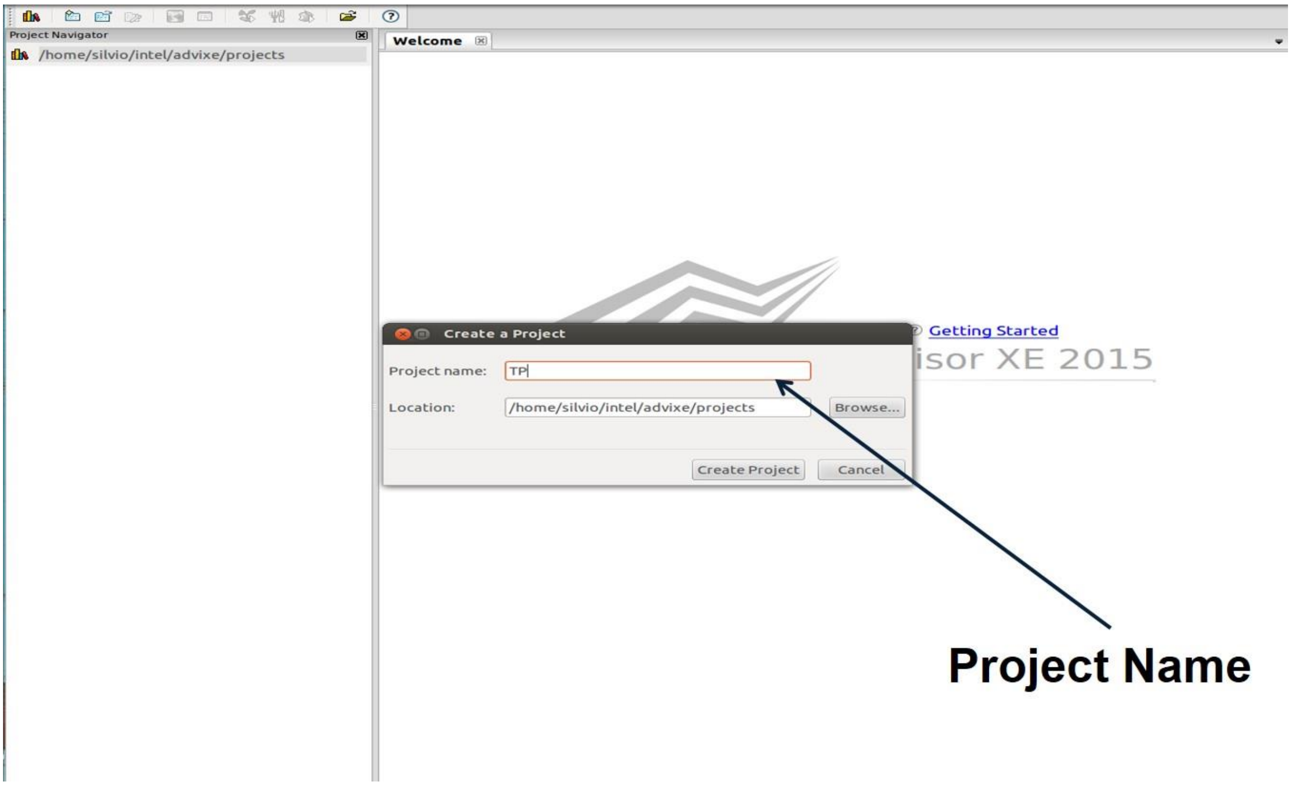
Intel Advisor – Create New Project

- Execute Intel Advisor on terminal: advixe-gui
- create new Advisor project:
  - name: Vectest
  - application: ~/handson/vec/VectorizationHandson
  - Source Folder: ~/handson/vec/



Welcome to Intel Advisor XE 2016

Vectorization Optimization and Thread Prototyping

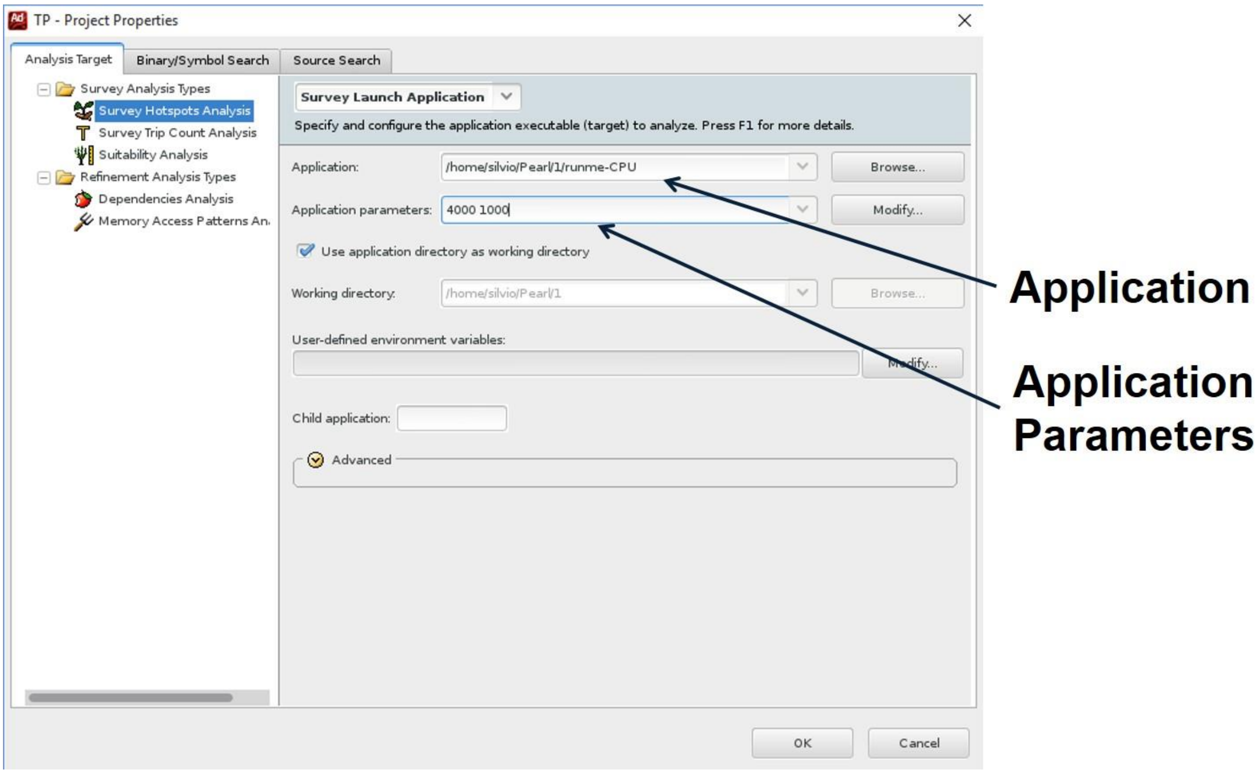


Project Navigator: /home/silvio/intel/advixe/projects

Project name: TP

Location: /home/silvio/intel/advixe/projects

Project Name



Survey Launch Application

Specify and configure the application executable (target) to analyze. Press F1 for more details.

Application: /home/silvio/Pearl/1/runme-CPU

Application parameters: 4000 1000

☑ Use application directory as working directory

Working directory: /home/silvio/Pearl/1

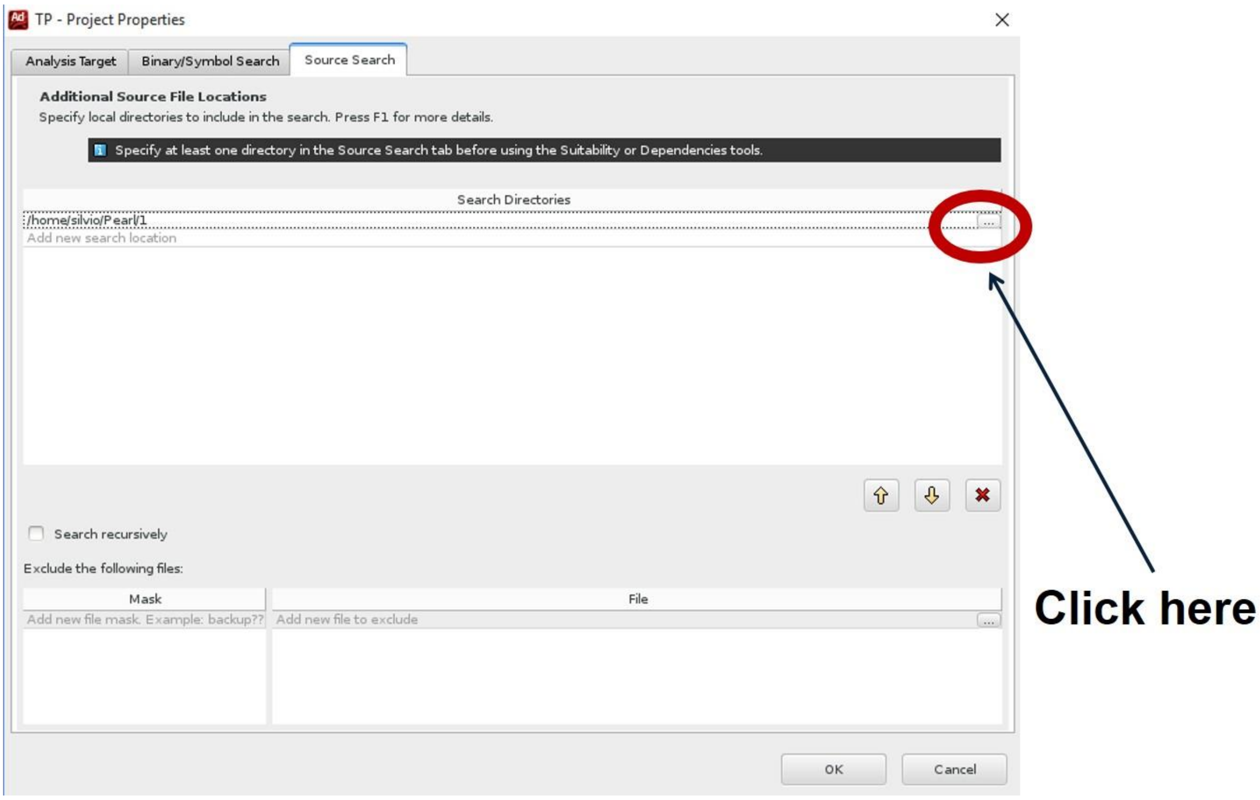
User-defined environment variables:

Child application:

Advanced

Application

Application Parameters



Additional Source File Locations

Specify local directories to include in the search. Press F1 for more details.

Specify at least one directory in the Source Search tab before using the Suitability or Dependencies tools.

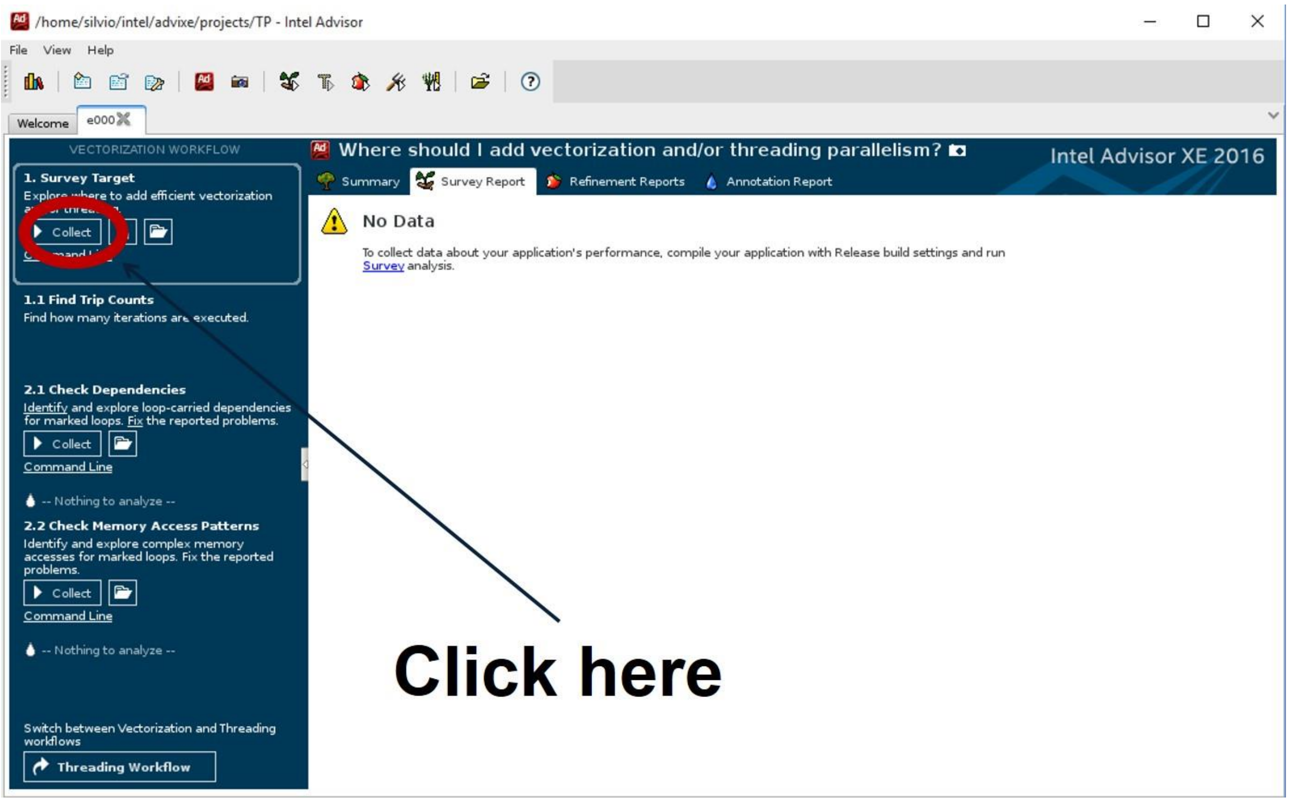
Search Directories
/home/silvio/Pearl/1
Add new search location

□ Search recursively

Exclude the following files:

Mask	File
Add new file mask. Example: backup??	Add new file to exclude

Click here


/home/silvio/intel/advixe/projects/TP - Intel Advisor
File View Help
Welcome
e000
VECTORIZATION WORKFLOW
1. Survey Target
Explore where to add efficient vectorization and/or threading
Collect
Command Line
1.1 Find Trip Counts
Find how many iterations are executed.
2.1 Check Dependencies
Identify and explore loop-carried dependencies for marked loops. Fix the reported problems.
Collect
Command Line
-- Nothing to analyze --
2.2 Check Memory Access Patterns
Identify and explore complex memory accesses for marked loops. Fix the reported problems.
Collect
Command Line
-- Nothing to analyze --
Switch between Vectorization and Threading workflows
Threading Workflow
Where should I add vectorization and/or threading parallelism?
Intel Advisor XE 2016
Summary
Survey Report
Refinement Reports
Annotation Report
No Data
To collect data about your application's performance, compile your application with Release build settings and run Survey analysis.
Click here

Change xhost

- Note that the code was compiled using SSE
- Recompile application using xhost and –g

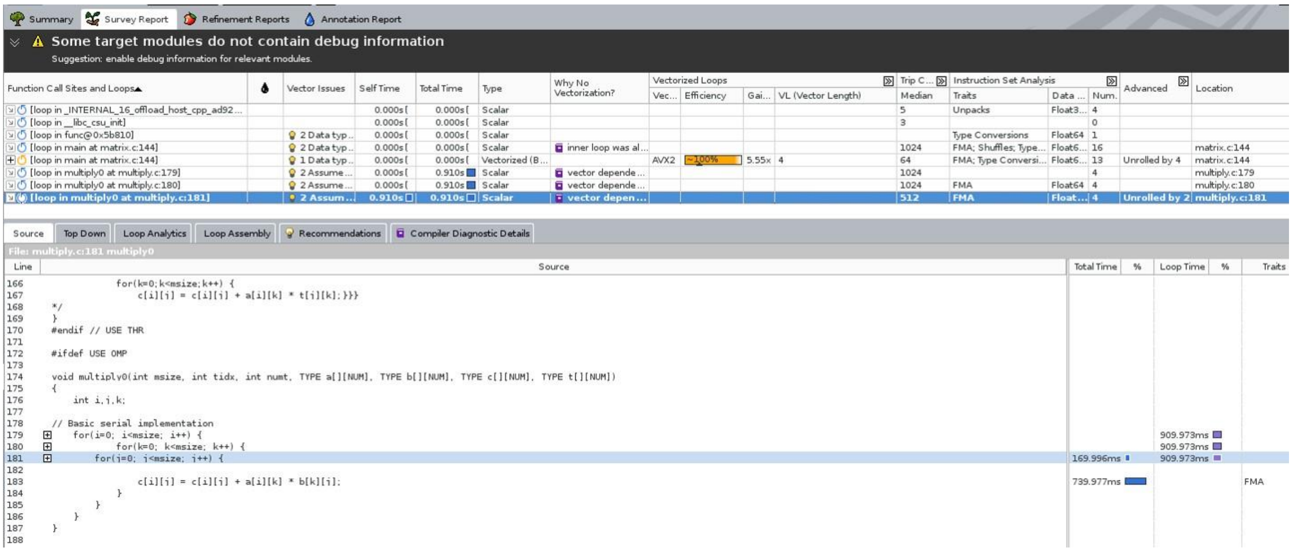
```
icc VectorizationHandson.c -o VectorizationHandson -vec-report6 –O3 –restrict –xhost -g
```

- Collect Survey Data again
- Note that now the code was compiled using AVX

Matrix Multiplication

- Compile code;
- Create new Advisor Project;
- Execute Survey Analysis;
- Execute Trip Count Analysis;
- Execute Check Dependencies Analysis;

- Compile code
  - Cd ~/handson/matrix/linux
  - Make clean ; make icc
- Execute Intel Advisor on terminal: advixe-gui
- create new Advisor project:
  - name: matrix-handson
  - application: ~/handson/matrix/linux/matrix.icc
  - Source Folder: ~/handson/matrix/src/
- Execute Survey Analysis



Some target modules do not contain debug information

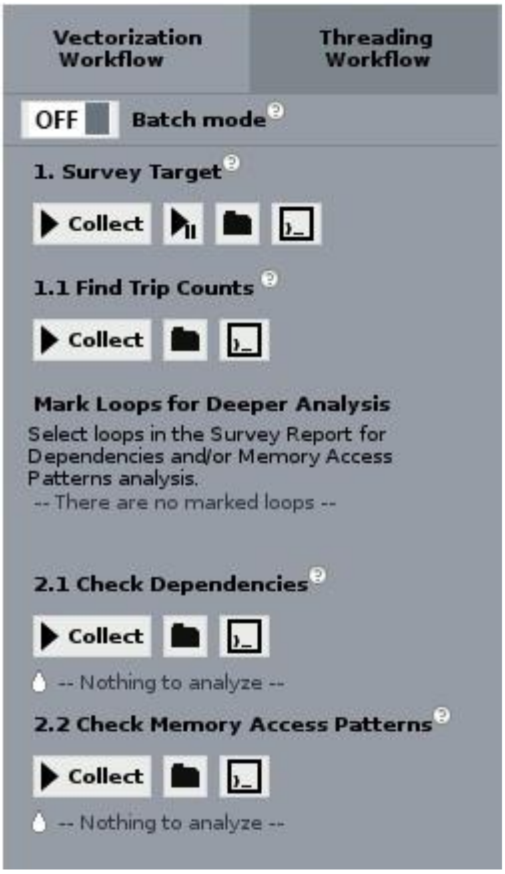
Suggestion: enable debug information for relevant modules.

Function Call Sites and Loops▲	💧	Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops				Trip C...	Instruction Set Analysis			Advanced	Location
							Vec...	Efficiency	Gai...	VL (Vector Length)	Median	Traits	Data ...	Num.		
[loop in _INTERNAL_16_offload_host_cpp_ad92...			0.000s	0.000s	Scalar						5	Unpacks	Float3...	4		
[loop in __libc_csu_init]			0.000s	0.000s	Scalar						3			0		
[loop in func@0x5b810]		2 Data typ...	0.000s	0.000s	Scalar							Type Conversions	Float64	1		
[loop in main at matrix.c:144]		2 Data typ...	0.000s	0.000s	Scalar	inner loop was al...					1024	FMA; Shuffles; Type...	Float6...	16		matrix.c:144
[loop in main at matrix.c:144]		1 Data typ...	0.000s	0.000s	Vectorized (B...		AVX2	~100%	5.55x	4	64	FMA; Type Conversi...	Float6...	13	Unrolled by 4	matrix.c:144
[loop in multiply0 at multiply.c:179]		2 Assume...	0.000s	0.910s	Scalar	vector depende...					1024			4		multiply.c:179
[loop in multiply0 at multiply.c:180]		2 Assume...	0.000s	0.910s	Scalar	vector depende...					1024	FMA	Float64	4		multiply.c:180
[loop in multiply0 at multiply.c:181]		**2 Assum...**	**0.910s**	**0.910s**	**Scalar**	**vector depen...**					**512**	**FMA**	**Float...**	**4**	**Unrolled by 2**	**multiply.c:181**

File: multiply.c:181 multiply0

```
Line  Source                                                                                                    Total Time   %   Loop Time   %   Traits
166               for(k=0;k<msize;k++) {
167                   c[i][j] = c[i][j] + a[i][k] * t[j][k];}}}
168       */
169       }
170       #endif // USE THR
171
172       #ifdef USE OMP
173
174       void multiply0(int msize, int tidx, int numt, TYPE a[][NUM], TYPE b[][NUM], TYPE c[][NUM], TYPE t[][NUM])
175       {
176           int i,j,k;
177
178       // Basic serial implementation
179   ⊞       for(i=0; i<msize; i++) {                                                                                      909.973ms
180   ⊞               for(k=0; k<msize; k++) {                                                                              909.973ms
181   ⊞           for(j=0; j<msize; j++) {                                                               169.996ms          909.973ms
182
183                   c[i][j] = c[i][j] + a[i][k] * b[k][j];                                              739.977ms                       FMA
184               }
185           }
186       }
187   }
188
```

- Note that Inner loop was not automatic vectorized.



1. Survey Target

1.1 Find Trip Counts

Mark Loops for Deeper Analysis

Select loops in the Survey Report for Dependencies and/or Memory Access Patterns analysis.

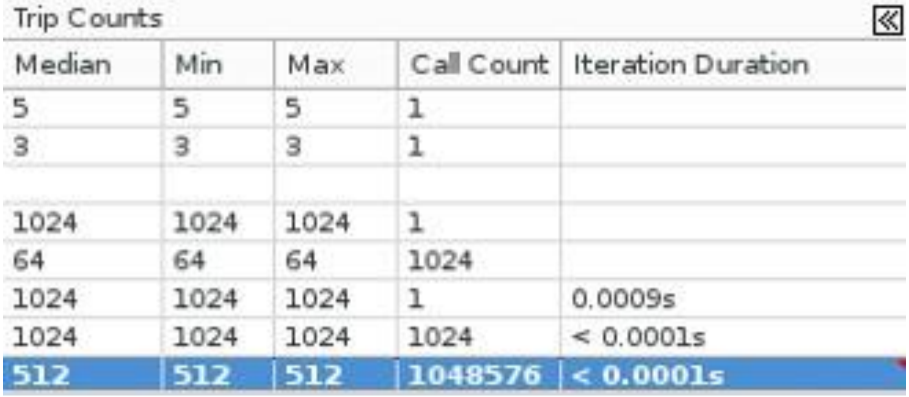
-- There are no marked loops --

2.1 Check Dependencies

-- Nothing to analyze --

2.2 Check Memory Access Patterns

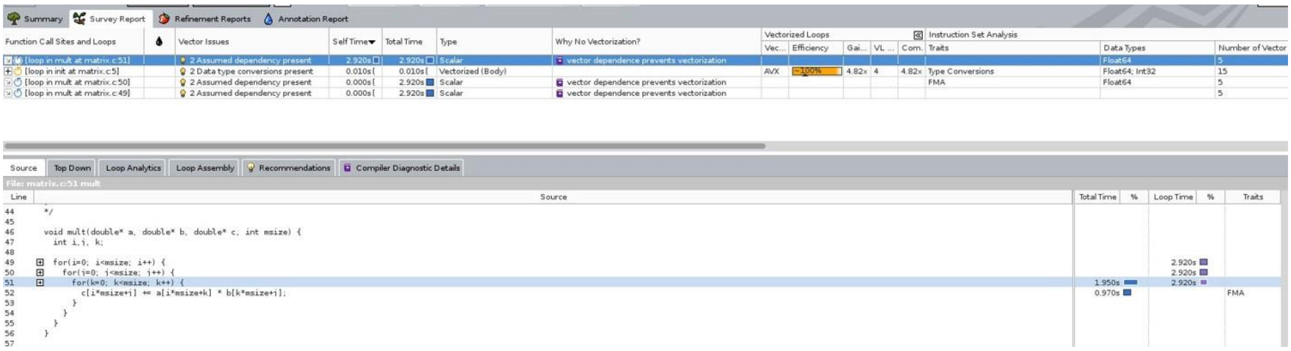
-- Nothing to analyze --



Trip Counts

Median	Min	Max	Call Count	Iteration Duration
5	5	5	1	
3	3	3	1	
1024	1024	1024	1	
64	64	64	1024	
1024	1024	1024	1	0.0009s
1024	1024	1024	1024	< 0.0001s
512	512	512	1048576	< 0.0001s

- Mark Inner Loop for deeper analysis;
- Click on “check dependency”;



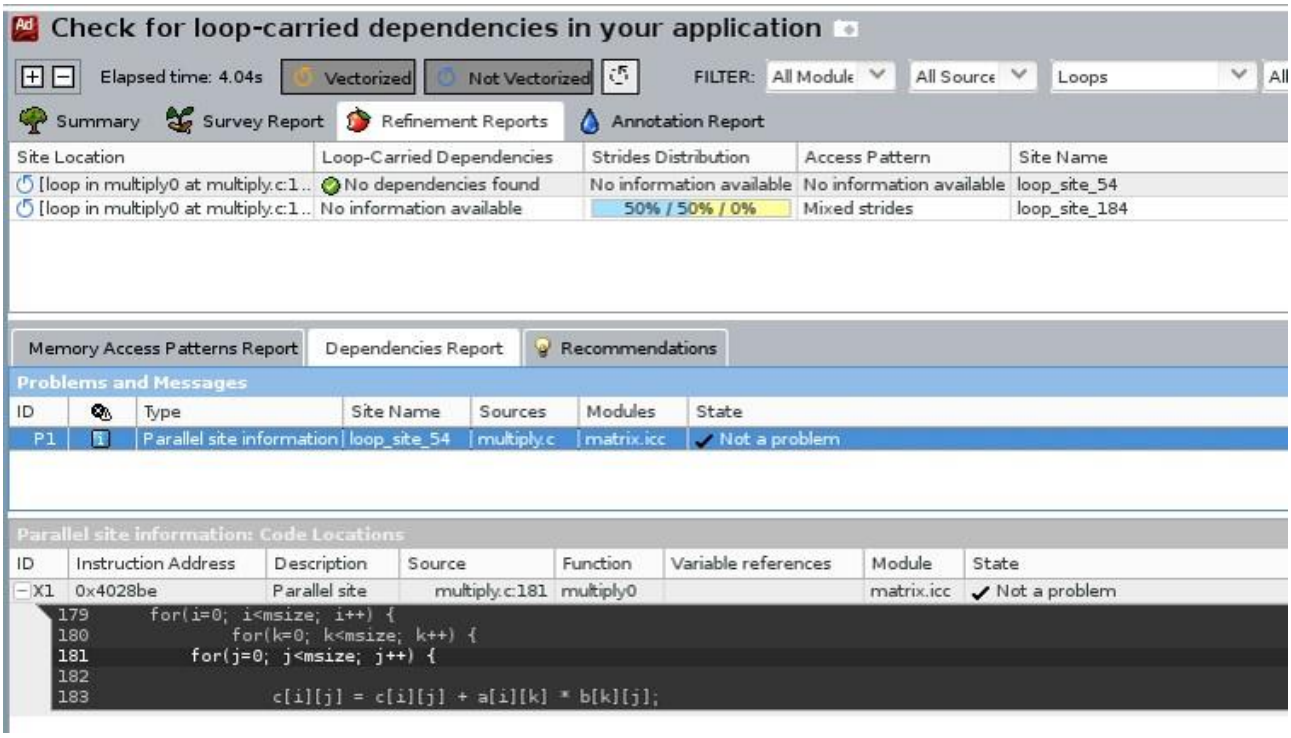
Function Call Sites and Loops		Vector Issues	Self Time▾	Total Time	Type	Why No Vectorization?	Vectorized Loops					Instruction Set Analysis		
							Vec...	Efficiency	Gai...	VL ...	Com.	Traits	Data Types	Number of Vector
[loop in mult at matrix.c:51]		2 Assumed dependency present	2.920s	2.920s	Scalar	vector dependence prevents vectorization							Float64	5
[loop in init at matrix.c:5]		2 Data type conversions present	0.010s	0.010s	Vectorized (Body)		AVX	~100%	4.82x	4	4.82x	Type Conversions	Float64; Int32	15
[loop in mult at matrix.c:50]		2 Assumed dependency present	0.000s	2.920s	Scalar	vector dependence prevents vectorization						FMA	Float64	5
[loop in mult at matrix.c:49]		2 Assumed dependency present	0.000s	2.920s	Scalar	vector dependence prevents vectorization								5

File: matrix.c:51 mult

```
 */

void mult(double* a, double* b, double* c, int msize) {
  int i,j, k;

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

Line	Total Time	%	Loop Time	%	Traits
49			2.920s		
50			2.920s		
51	1.950s		2.920s		
52	0.970s				FMA

- No dependencies found. It is safe to vectorize;



Check for loop-carried dependencies in your application

Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
[loop in multiply0 at multiply.c:1..	No dependencies found	No information available	No information available	loop_site_54
[loop in multiply0 at multiply.c:1..	No information available	50% / 50% / 0%	Mixed strides	loop_site_184

Problems and Messages

ID		Type	Site Name	Sources	Modules	State
P1		Parallel site information	loop_site_54	multiply.c	matrix.icc	✓ Not a problem

Parallel site information: Code Locations

ID	Instruction Address	Description	Source	Function	Variable references	Module	State
X1	0x4028be	Parallel site	multiply.c:181	multiply0		matrix.icc	✓ Not a problem

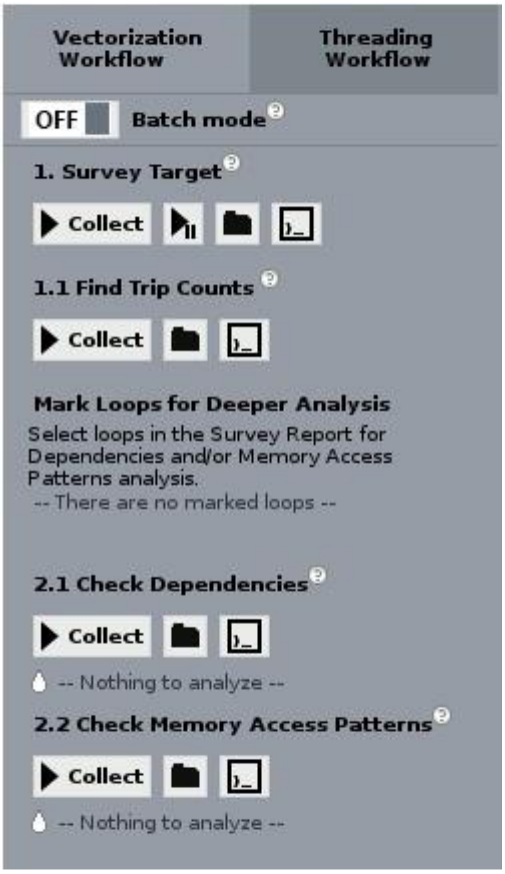
```
    for(i=0; i<msize; i++) {
            for(k=0; k<msize; k++) {
        for(j=0; j<msize; j++) {

                c[i][j] = c[i][j] + a[i][k] * b[k][j];
```

- Put #pragma simd in top of inner loop on function multiply0:
  - Nano ~/handson/matrix/src/multiply.c

```
for(i=0; i<msize; i++) {
   for(k=0; k<msize; k++) {
      #pragma simd
 for(j=0; j<msize; j++) {
```

- Recompile application
  - Make clean ; make icc
  - Run survey data again;
- Note that inner loop is vectorized now;

- Run Check Memory Access Patterns;



1. Survey Target

1.1 Find Trip Counts

Mark Loops for Deeper Analysis

Select loops in the Survey Report for Dependencies and/or Memory Access Patterns analysis.

-- There are no marked loops --

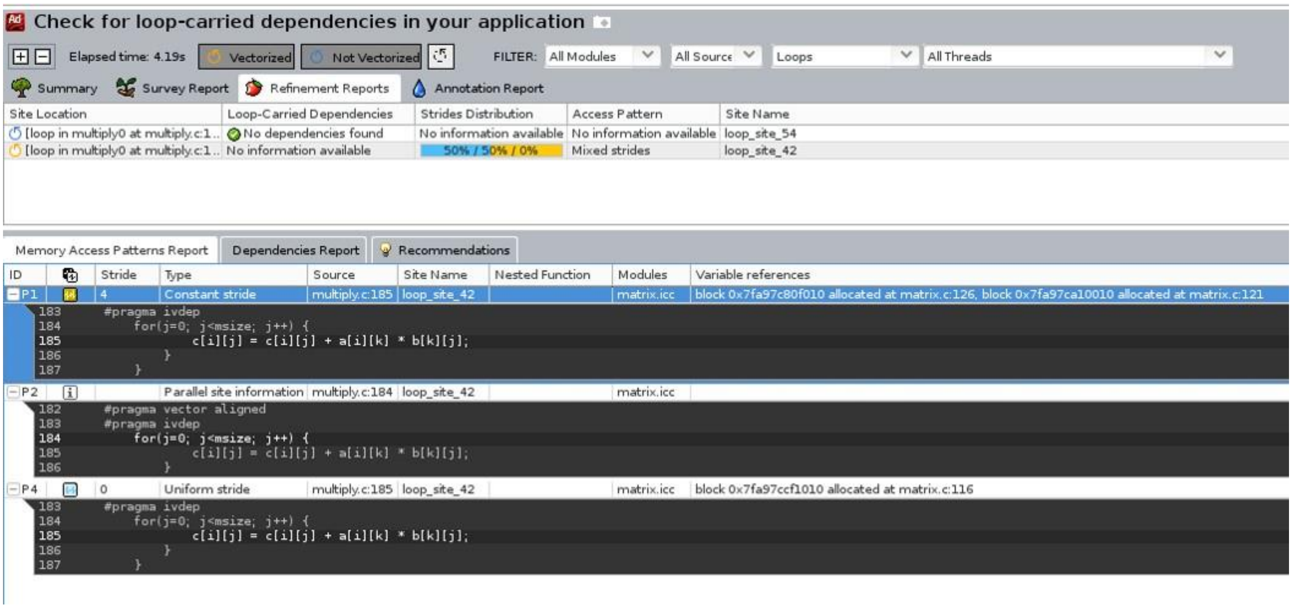
2.1 Check Dependencies

-- Nothing to analyze --

2.2 Check Memory Access Patterns

-- Nothing to analyze --

- Memory access Patterns Results:
  - 50% constant stride;
  - 50% unit stride;



Site Location	Loop-Carried Dependencies	Strides Distribution	Access Pattern	Site Name
[loop in multiply0 at multiply.c:1..	No dependencies found	No information available	No information available	loop_site_54
[loop in multiply0 at multiply.c:1..	No information available	50% / 50% / 0%	Mixed strides	loop_site_42

ID		Stride	Type	Source	Site Name	Nested Function	Modules	Variable references
P1		4	Constant stride	multiply.c:185	loop_site_42		matrix.icc	block 0x7fa97c80f010 allocated at matrix.c:126, block 0x7fa97ca10010 allocated at matrix.c:121

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

ID		Stride	Type	Source	Site Name	Nested Function	Modules	Variable references
P2	i		Parallel site information	multiply.c:184	loop_site_42		matrix.icc	

```
#pragma vector aligned
#pragma ivdep
    for(j=0; j<msize; j++) {
        c[i][j] = c[i][j] + a[i][k] * b[k][j];
    }
```

ID		Stride	Type	Source	Site Name	Nested Function	Modules	Variable references
P4		0	Uniform stride	multiply.c:185	loop_site_42		matrix.icc	block 0x7fa97ccf1010 allocated at matrix.c:116

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