Introdução à Vetorização em Arquiteturas Paralelas Híbridas

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

WSCAD 2016
06 October 2016

Agenda

- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- Intel Architectures;
- Vectorization Process;
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization;
- Examples.

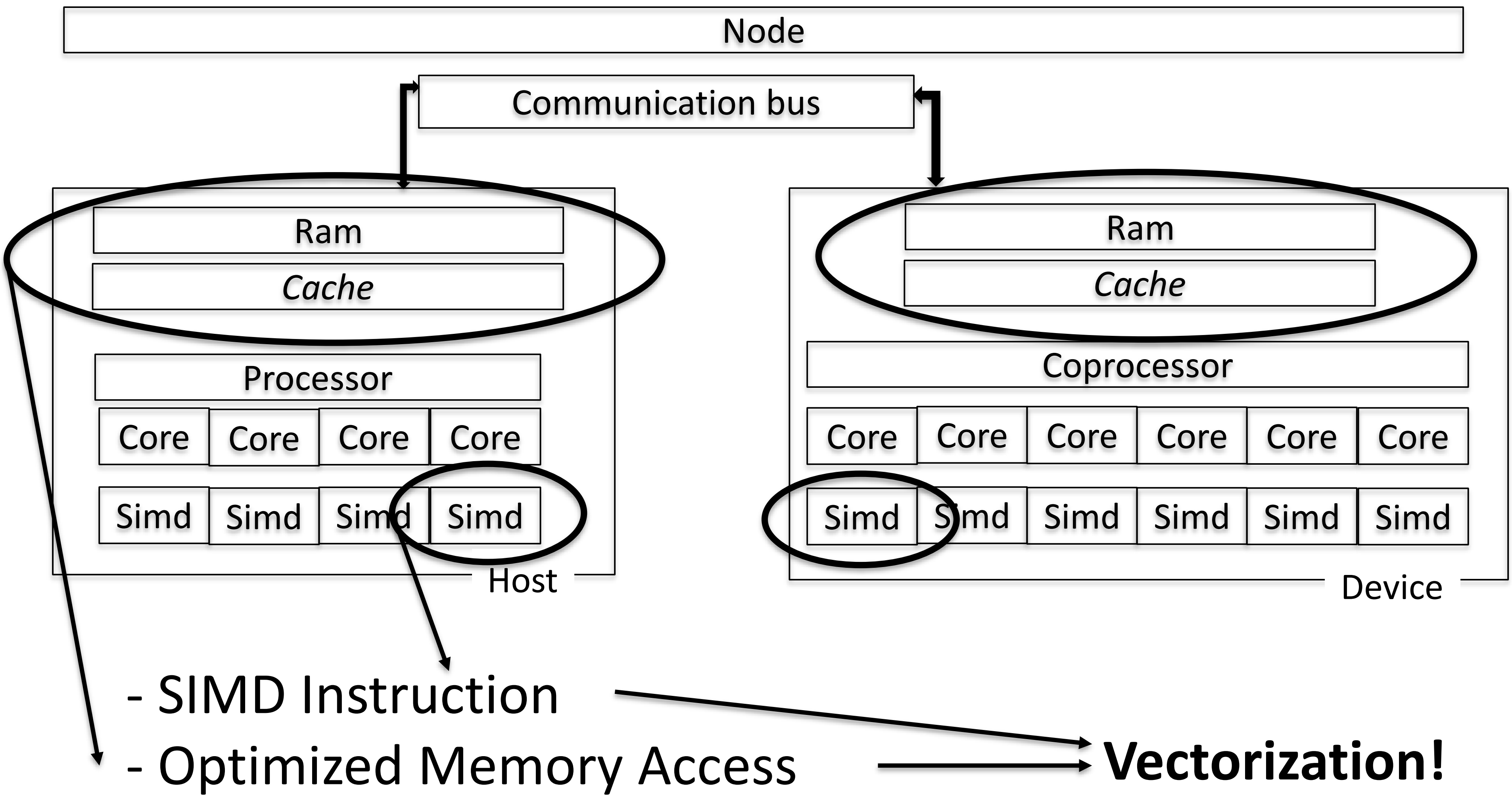
Agenda

- **Hybrid Parallel Architectures;**
- Memory System and Vector Processing Units;
- Intel Architectures;
- Vectorization Process;
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization;
- Examples.

Hybrid Parallel Architectures

- Heterogeneous computational systems:
  - Multicore processors;
  - Multi-level memory sub-system;
  - Input and Output sub-system;
- Multi-level parallelism:
  - Processing core;
  - Chip multiprocessor;
  - Computing node;
  - Computing cluster;
- Hybrid Parallel architectures
  - Coprocessors and accelerators;

Hybrid Parallel Architectures



Don’t use a single thread or vector lane

Exploiting the parallel universe

Instruction Level Parallelism

- Single thread (ST) performance
- Automatically exposed by HW/tools
- Effectively limited to a few instructions

Data Level Parallelism

- Single thread (ST) performance
- Exposed by tools and programming models
- Operate on 4/8/16 elements at a time

Task Level Parallelism

- Multi thread/task (MT) performance
- Exposed by programming models
- Execute tens/hundreds/thousands task concurrently

Process Level Parallelism

- Multi Process (MP) performance
- Exposed by programming models
- Execute tens/hundreds/thousands of process concurrently across several nodes

Agenda

- Hybrid Parallel Architectures;
- **Memory System and Vector Processing Units;**
- Intel Architectures;
- Vectorization Process;
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization;
- Examples.

Memory System

Fast

CPU Register: internal Processor Memory. Stores data or instruction to be executed

Cache: stores segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations

Main memory: only program and data currently needed by the processor resides in main memory

Auxiliary memory: devices that provides backup storage

Larger in Size

Cache Memory

- Cache Memory is employed in computer systems to compensate for the difference in speed between main memory access time and processor.

- Operating System controls the load of Data to Cache;
  - such load can be guided by the developer

- The performance of cache memory is frequently measured in terms of hit ratio.

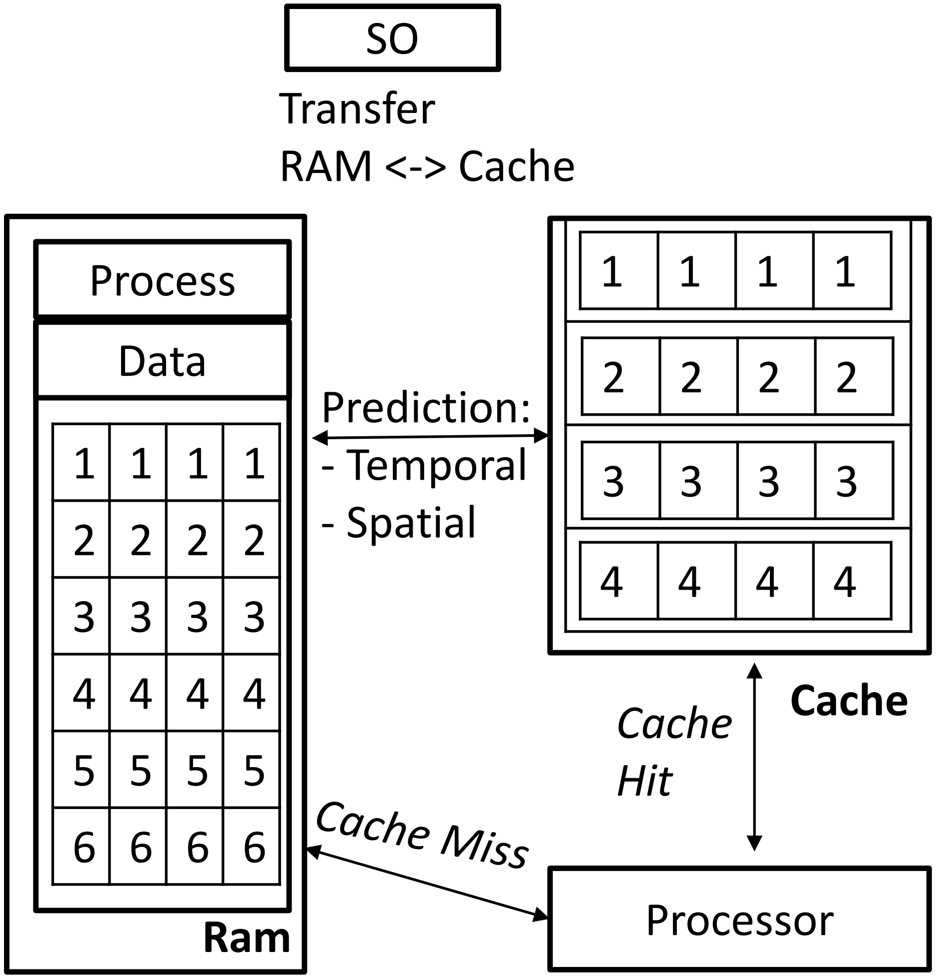
  - When the CPU refers to memory and finds the word in cache, it is said to produce a **hit**.

  - If the word is not found in cache, it is in main memory and it counts as a **miss**

Locality

- Temporal locality: if an item was referenced, it will be referenced again soon (e.g. cyclical execution in loops);

- Spatial locality: if an item was referenced, items close to it will be referenced too (the very nature of every program – serial stream of instructions)



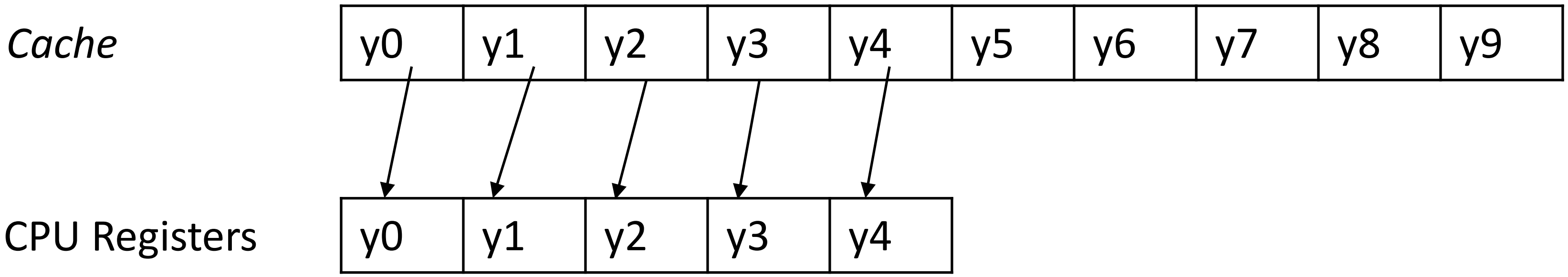
Stride (array elements)

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

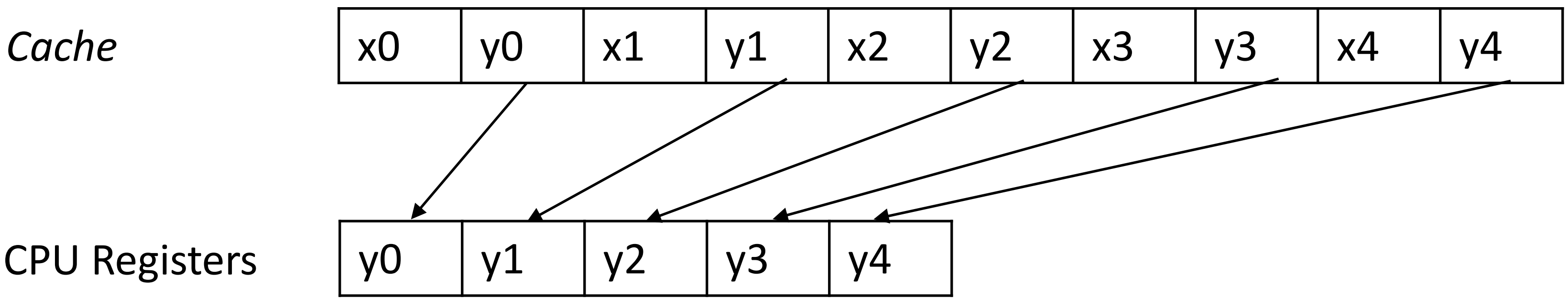
- Strided access with stride k means touching every kth memory element
  - Unit Stride :
    - ❑ Sequential access (0, 1, 2, 3, 4, 5, 6, ...)
  - Non-unit stride
    - ❑ Constant Stride =
      - ○ 2 is (0, 2, 4, 6, 8, ...)
    - ❑ k is (0, k, 2k, 3k, 4k, ...)
    - ❑ Random Access;

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

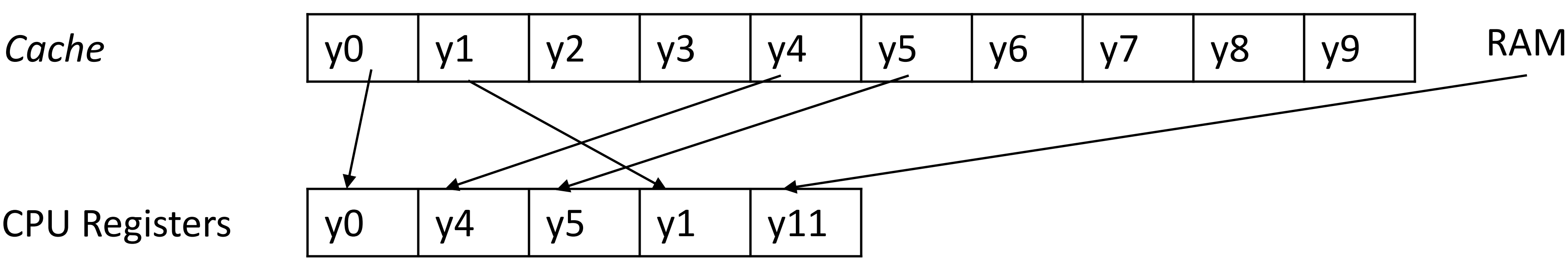
Unit-Stride



Constant-Stride



Random-Stride



Vectorization

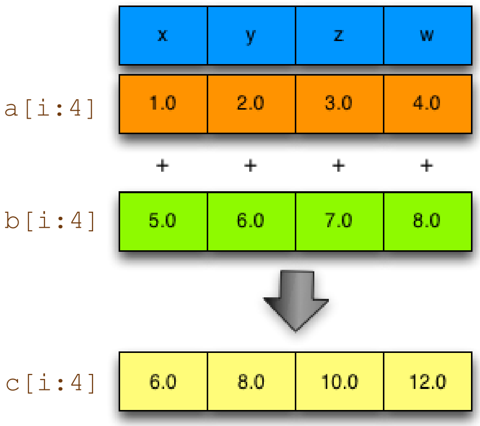
- Vectorization
  - Loading data into cache accordingly;
  - Store elements on SIMD registers or vectors;
  - Apply the same operation to a set of Data at the same time;
  - Iterations need to be independent;
  - Usually on inner loops.

Scalar loop

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

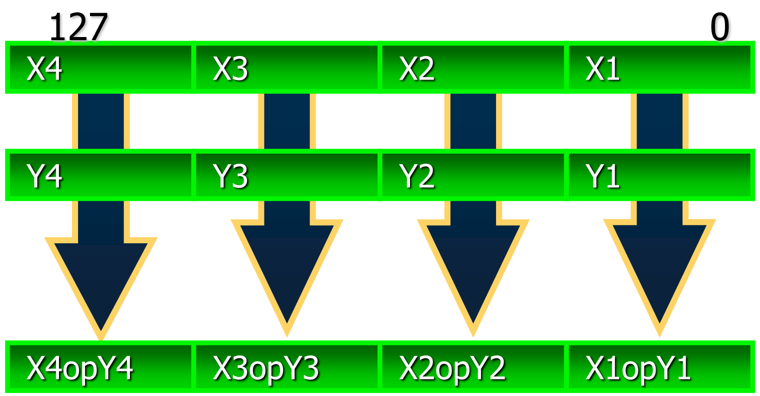
SIMD loop (4 elements)

```
for (int i = 0; i < N; i += 4)
    c[i:4] = a[i:4] + b[i:4];
```



Vectorial processing Intel® Architecture

MMX	Intel® SSE	Intel® AVX / AVX2	Intel® MIC / AVX-512
Vector size: 64bit Data types: 8, 16 and 32 bit integers VL: 2,4,8	Vector size: 128bit Data types: 8,16,32,64 bit integers 32 and 64bit float VL: 2,4,8,16	Vector size: 256bit Data types: 32 and 64 bit floats VL: 4, 8, 16 Sample: Xi, Yi 32 bit int or float	Vector size: 512bit Data types: 32 and 64 bit integers 32 and 64bit float VL: 8,16

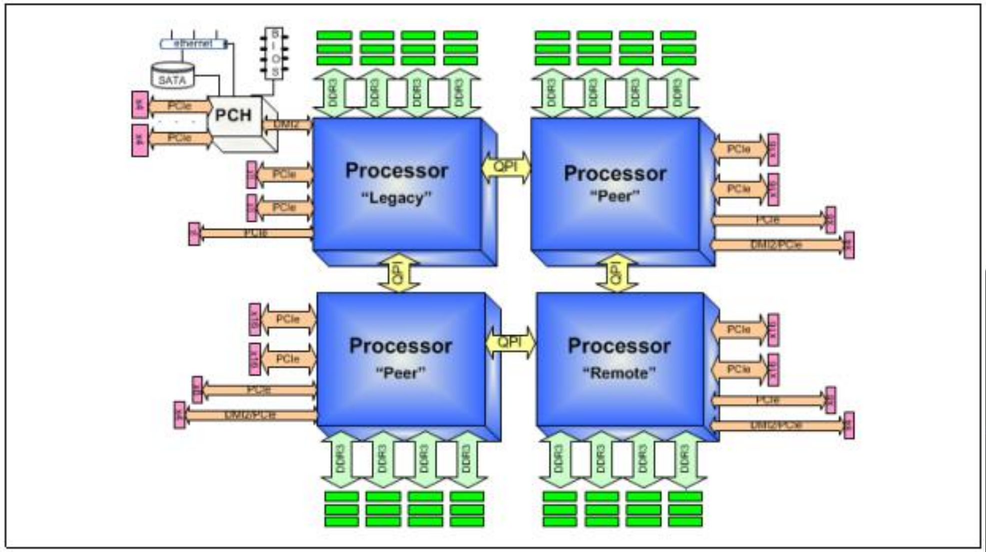


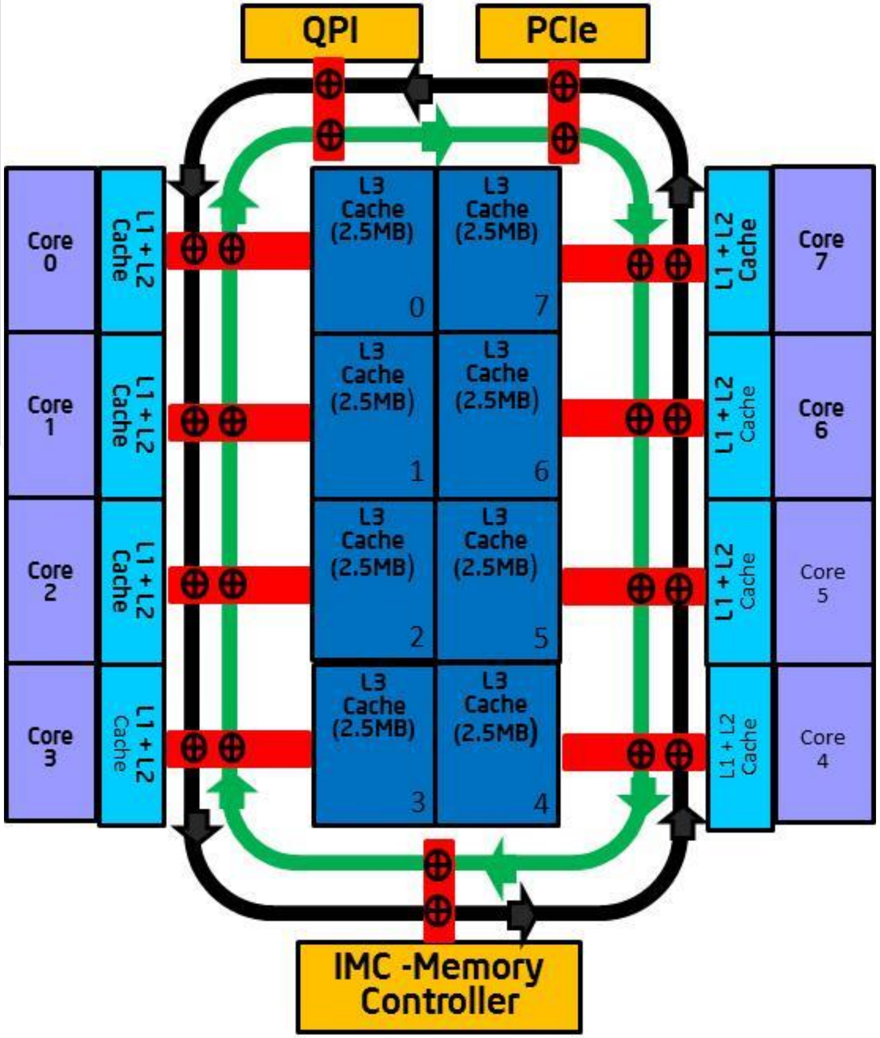
Exponential & Reciprocal Instructions (ERI)
Prefetch Instructions (PFI)
Foundation instructions (FI)
Conflict Detection Instructions (CDI)
Byte & Word Instructions (BWI)
Double-/Quad-word Instructions (DQI)
Vector Length Extensions (VLE)

Agenda

- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- **Intel Architectures;**
- Vectorization Process;
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization;
- Examples.

Intel Xeon Architecture Overview



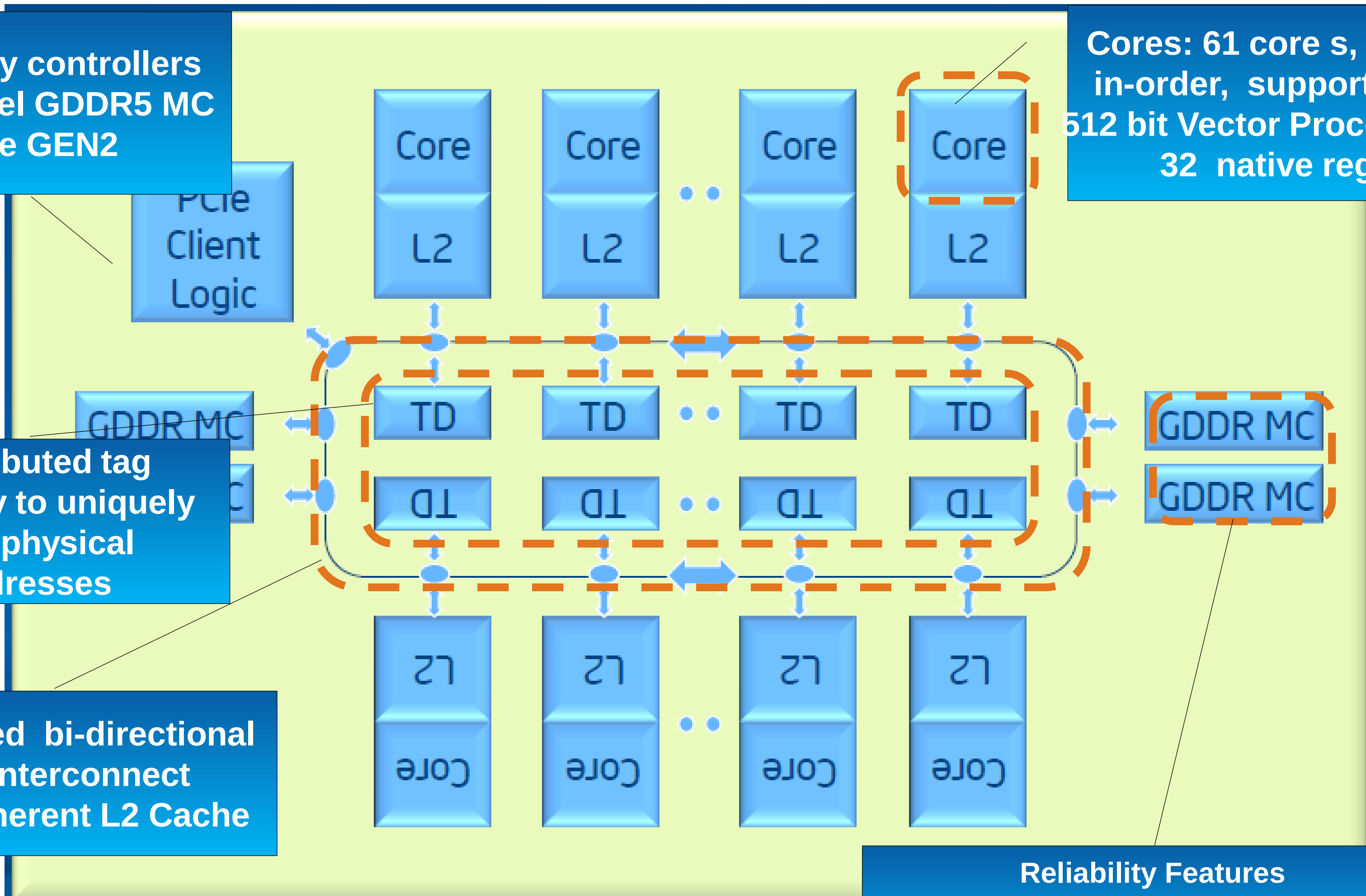


•Socket: mechanical component that provides mechanical and electrical connections between a microprocessor and a printed circuit board (PCB).

•QPI (Intel QuickPath Interconnect): high speed, packetized, point-to-point interconnection, that stitch together processors in distributed shared memory and integrated I/O platform architecture.

Intel® Xeon Phi™ Architecture Overview

- Knights Core (KNC)



8 memory controllers
16 Channel GDDR5 MC
PCIe GEN2

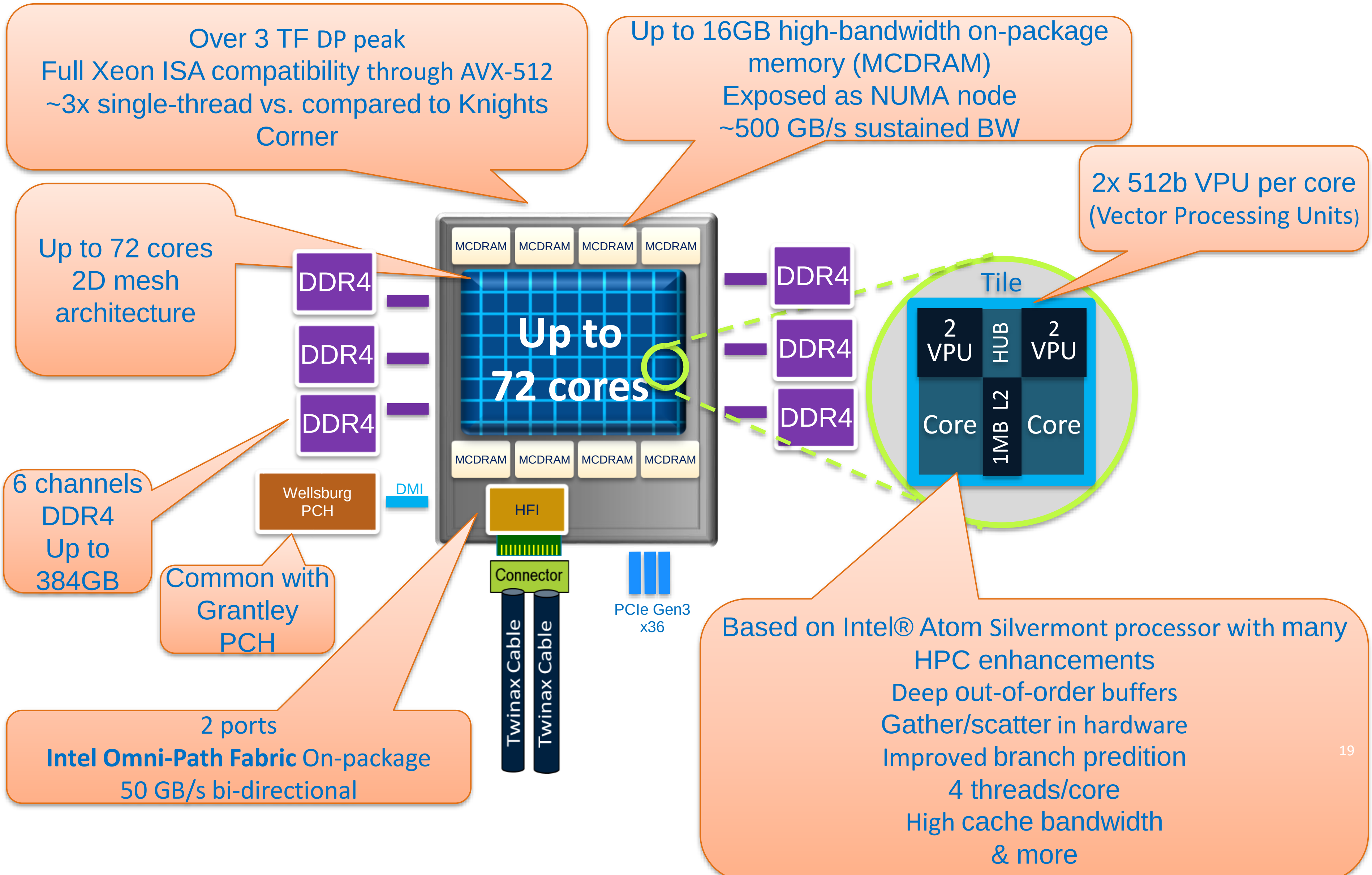
Cores: 61 core s, at 1.1 GHz
in-order, support 4 threads
512 bit Vector Processing Unit
32 native registers

Distributed tag directory to uniquely map physical addresses

High-speed bi-directional ring interconnect
Fully Coherent L2 Cache

Reliability Features
Parity on L1 Cache, ECC on memory
CRC on memory IO, CAP on memory IO

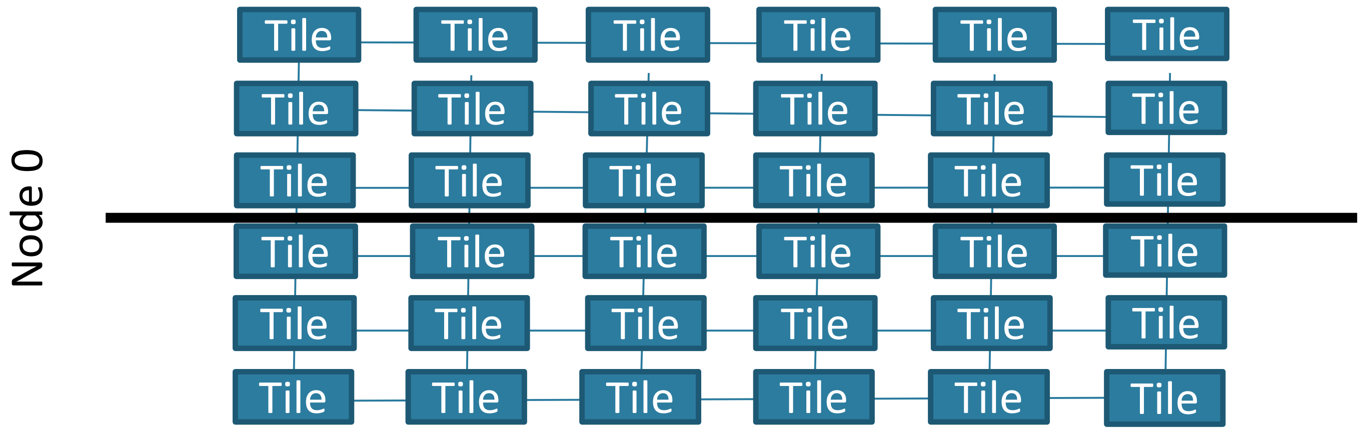
Knights Landing (KNL)



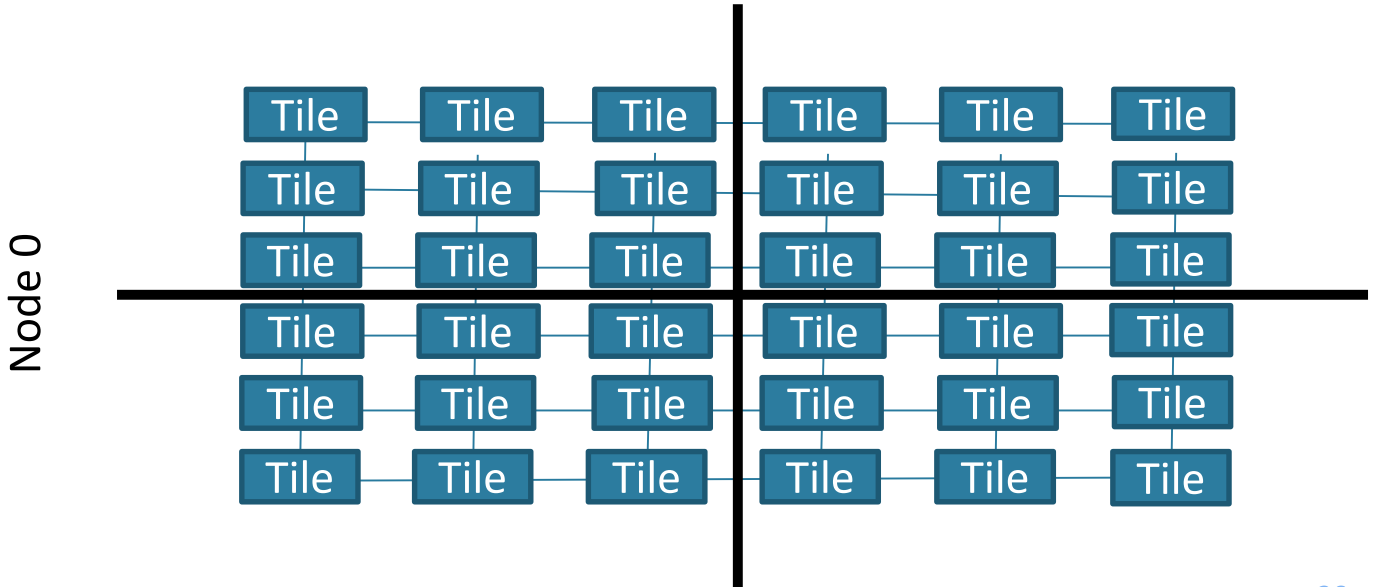
Cluster modes

One single space address

Hemisphere:
the tiles are divided
into two parts
called hemisphere



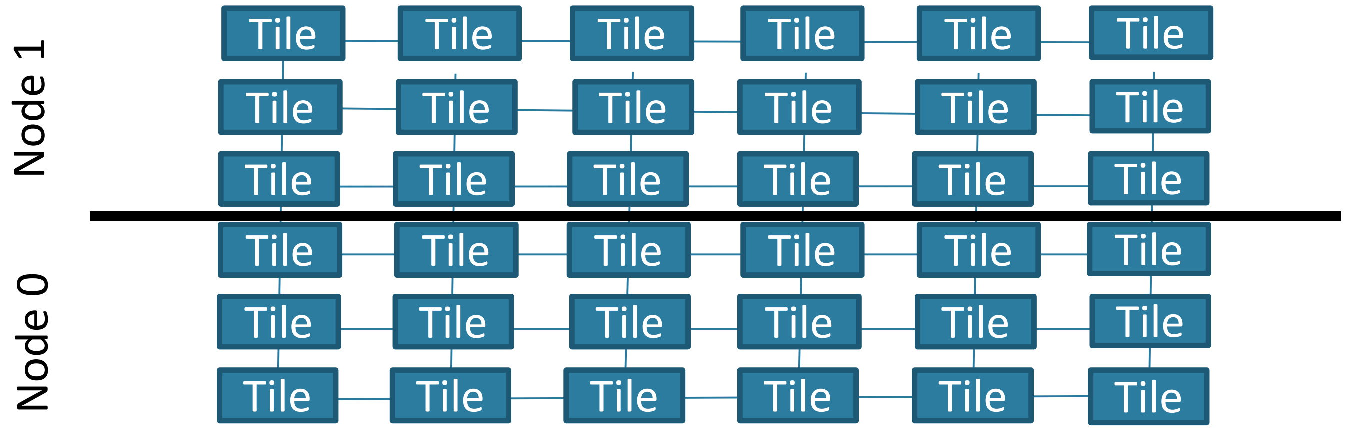
Quadrant:
tiles are divided
into two parts
called hemisphere
or into four parts
called qudrants



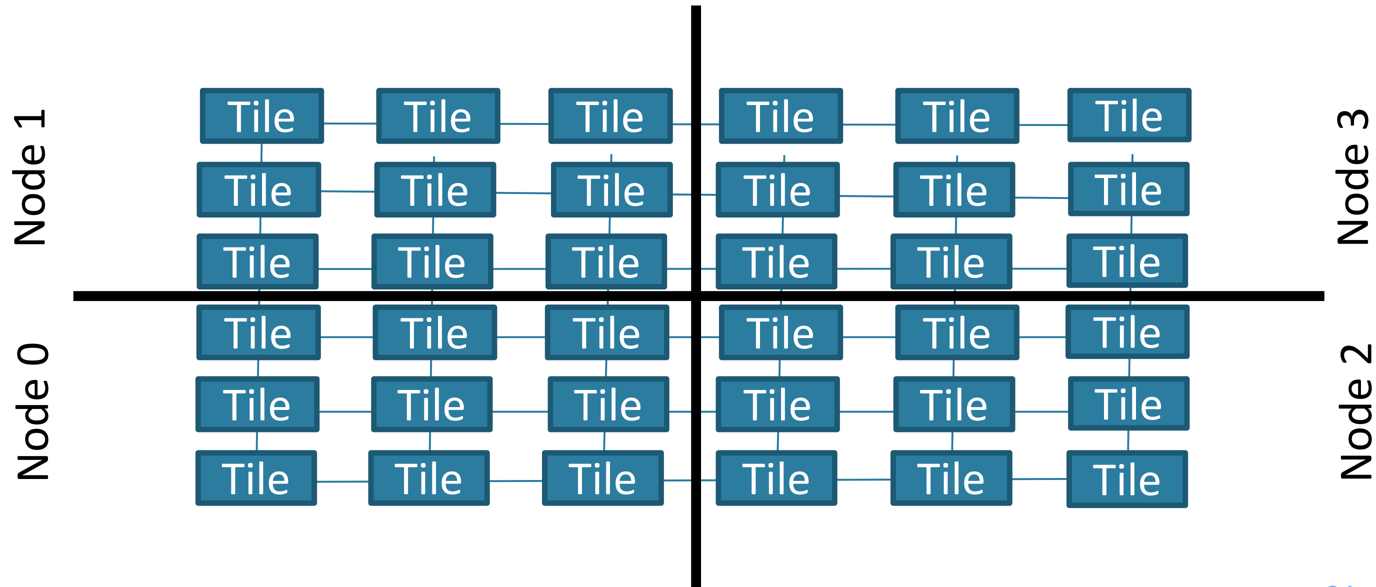
Cluster modes

Cache data are isolated in each sub numa domain

SNC-2:
the tiles are divided into two Numa Nodes

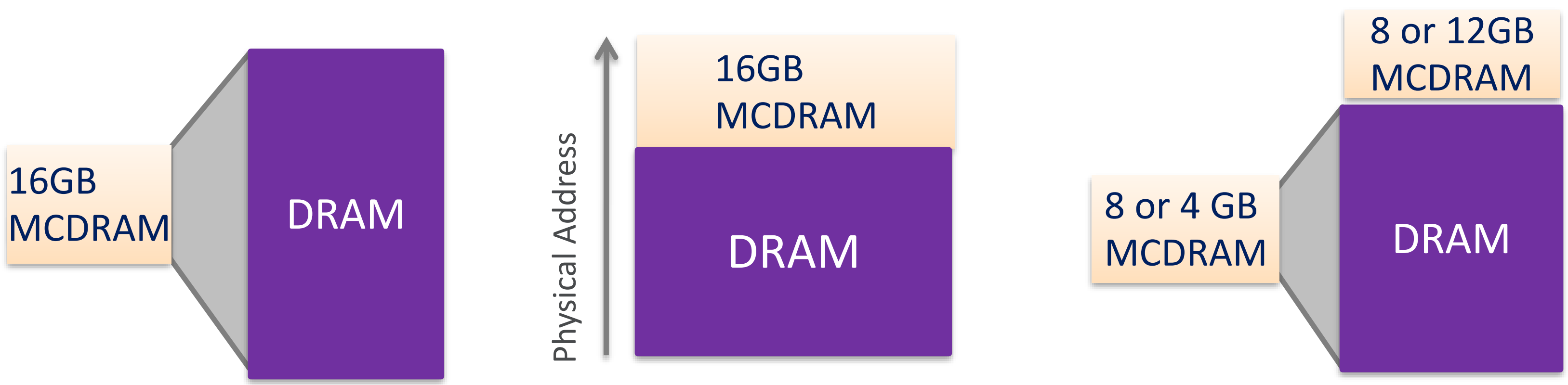


SNC-4:
the tiles are divided into two Numa Nodes



Integrated On-Package Memory Usage Models

Integrated On-Package Memory Usage Models



Split Options:
25/75% or 50/50%

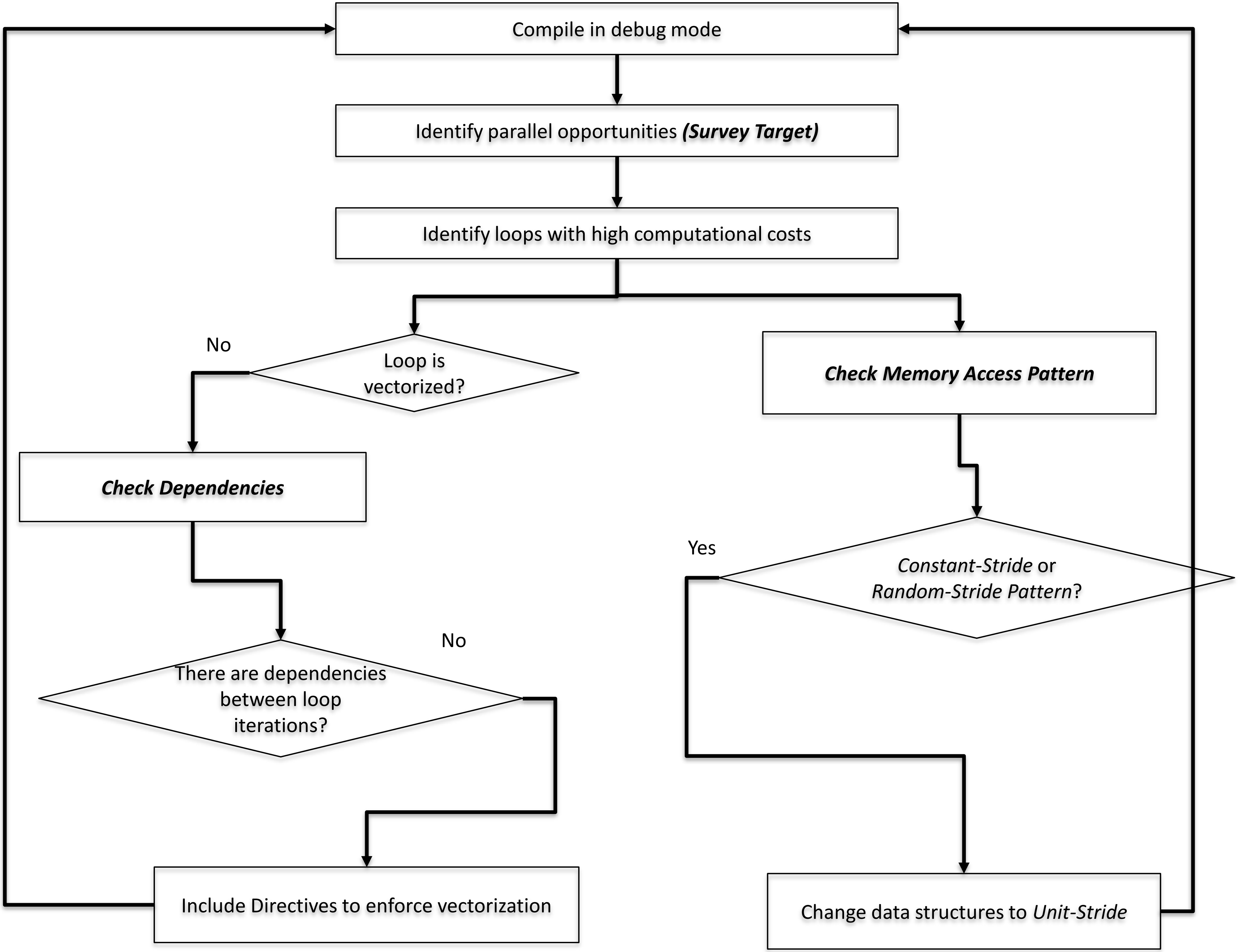
Cache Model	Flat Model	Hybrid Model
Hardware automatically manages the MCDRAM as a “L3 cache” between CPU and ext DDR memory	Manually manage how the app uses the integrated on-package memory and external DDR for peak perf	Harness the benefits of both Cache and Flat models by segmenting the integrated on-package memory
▪ App and/or data set is very large and will not fit into MCDRAM ▪ Unknown or unstructured memory access behavior	▪ App or portion of an app or data set that can be, or is needed to be “locked” into MCDRAM so it doesn’t get flushed out	▪ Need to “lock” in a relatively small portion of an app or data set via the Flat model ▪ Remaining MCDRAM can then be configured as Cache

Agenda

- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- Intel Architectures;
- **Vectorization Process;**
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization;
- Examples.

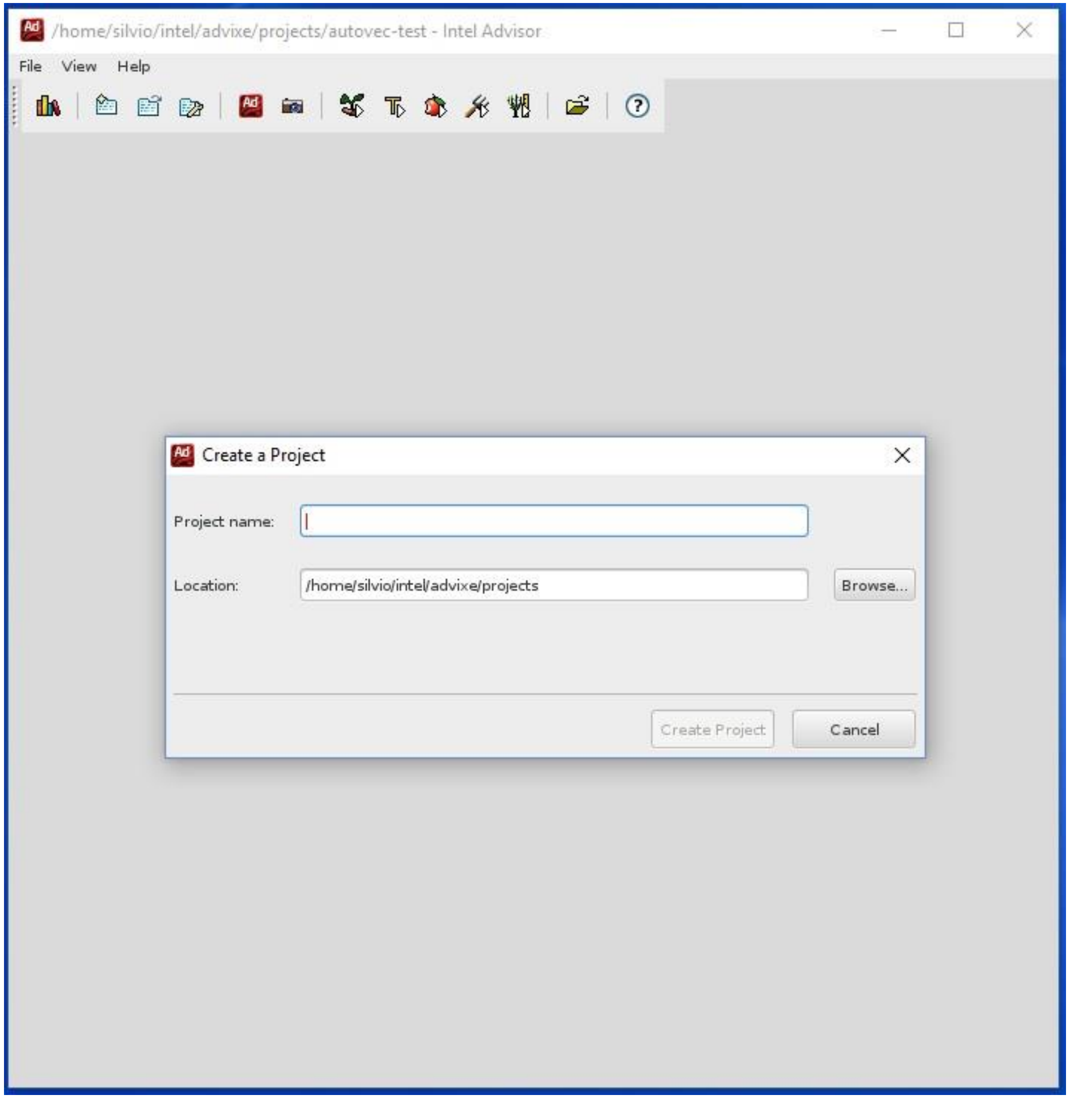
Intel Advisor

- 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.

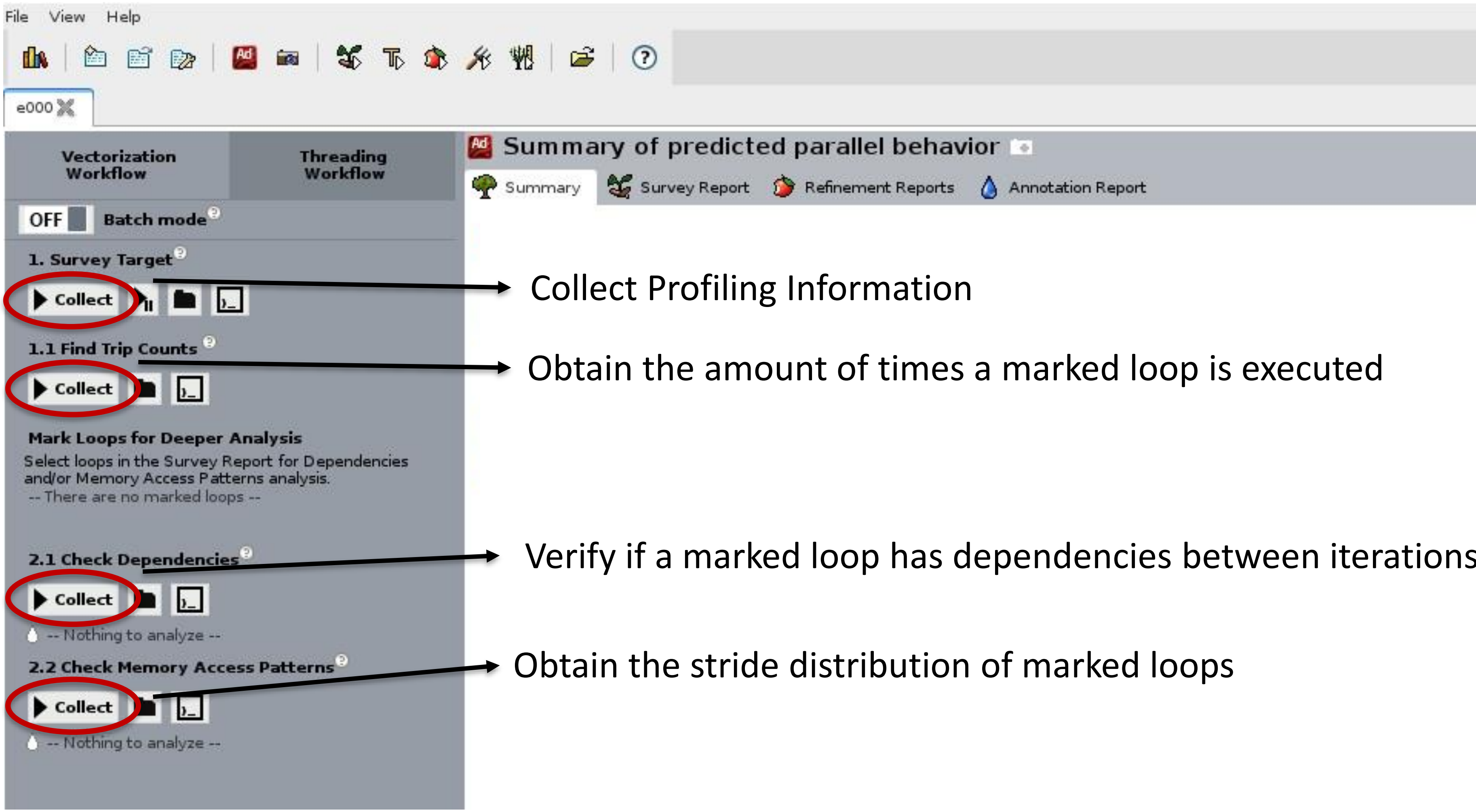


Compile in debug mode
Identify parallel opportunities *(Survey Target)*
Identify loops with high computational costs
Loop is vectorized?
No
Check Memory Access Pattern
Check Dependencies
Yes
Constant-Stride or Random-Stride Pattern?
No
There are dependencies between loop iterations?
Include Directives to enforce vectorization
Change data structures to Unit-Stride

Advisor – New Project



Advisor - Analysis



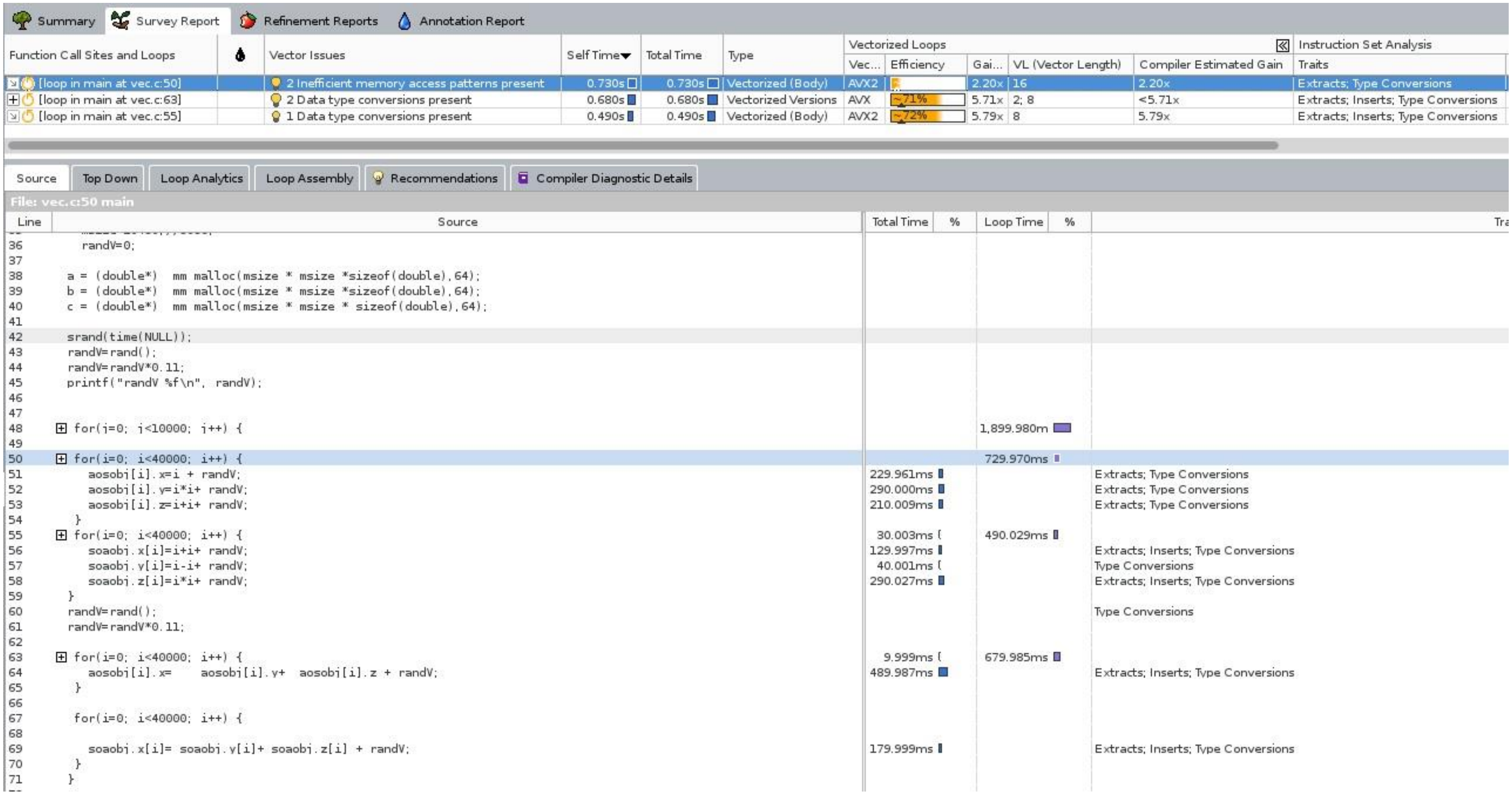
Collect Profiling Information

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

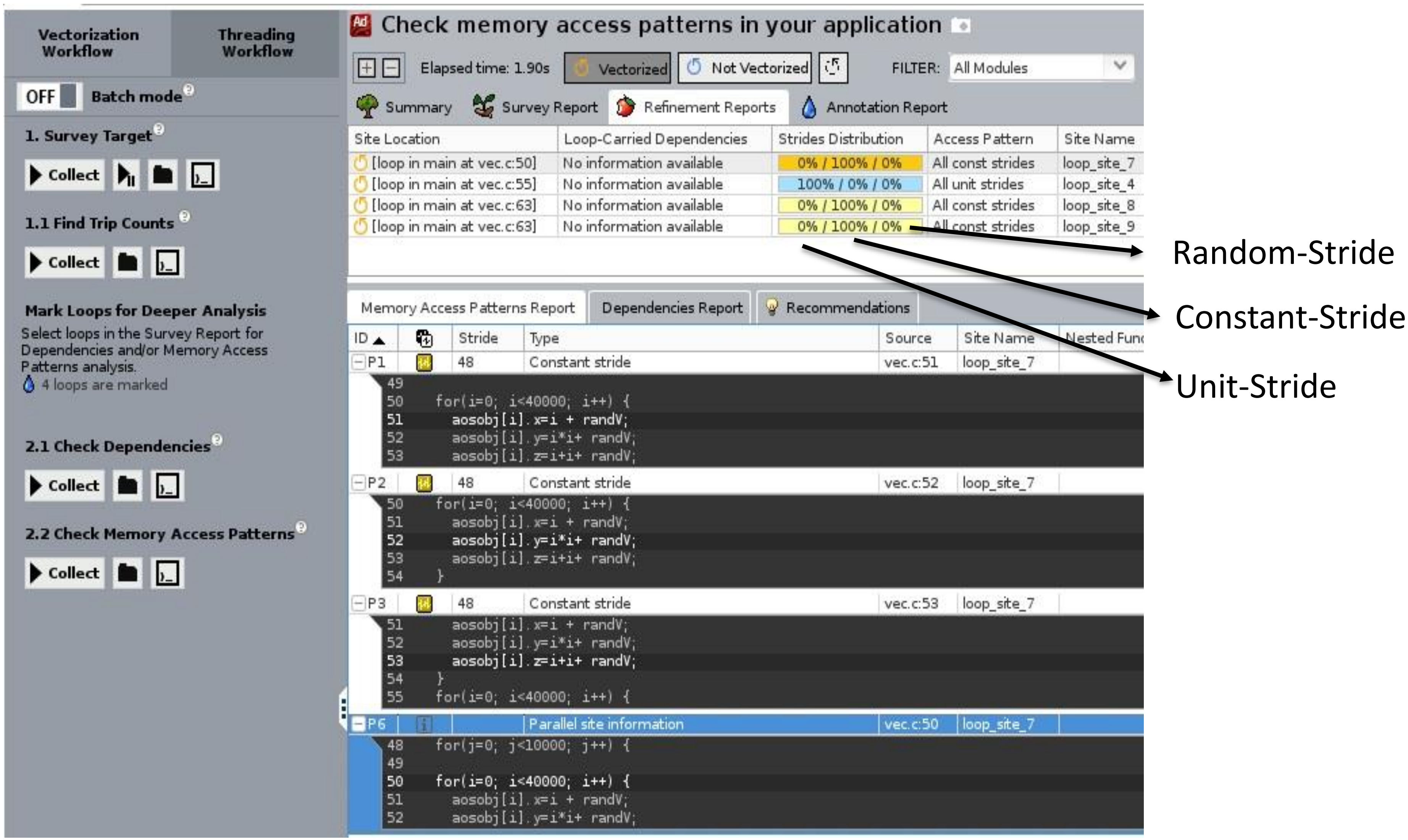


Advisor – Survey Target

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



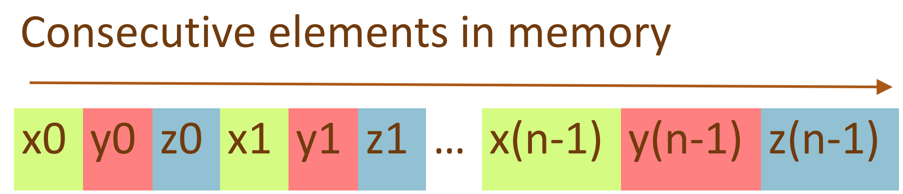
Agenda

- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- Intel Architectures;
- Vectorization Process;
- **Optimizing Memory Access;**
- Auto Vectorization;
- Guided Vectorization;
- Examples.

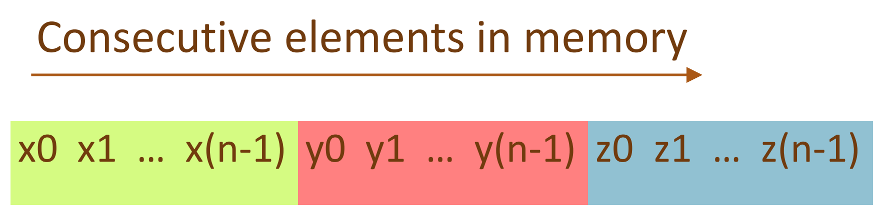
Data layout

- 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);
```




```
// 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]);
```



Agenda

- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- Intel Architectures;
- Vectorization Process;
- Optimizing Memory Access;
- **Auto Vectorization;**
- Guided Vectorization;
- Examples.

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.
-x*feature* -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.**
-ax*feature*	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

- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- Intel Architectures;
- Vectorization Process;
- Optimizing Memory Access;
- Auto Vectorization;
- **Guided Vectorization;**
- Examples.

Intel® compiler directives for vectorization

Directive	Clause	Description
`ivdep`		Instructs the compiler to ignore assumed vector dependencies.
`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

Agenda

- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- Intel Architectures;
- Vectorization Process;
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization;
- **Examples.**

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 ...	

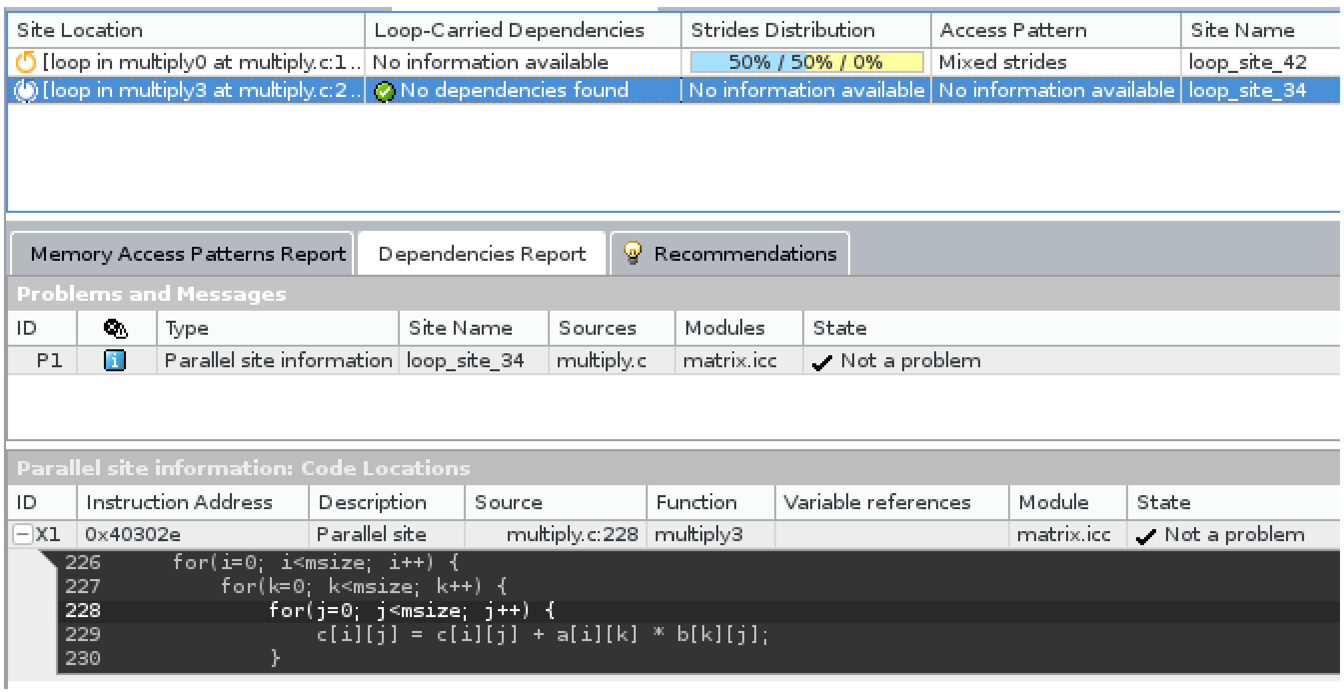
Source | Top Down | Loop Analytics | Loop Assembly | Recommendations | Compiler Diagnostic Details

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

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops		
							Vec...	Efficiency	Gain E...
[loop in multiply4 at multiply.c:245]			0.460s	0.460s	Vectorized (B...		AVX2	~100%	4.24x
[loop in multiply4 at multiply.c:243]			0.010s	0.470s	Scalar				
[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]		3 Data typ...	0.000s	0.000s	Scalar				
[loop in func@0x54bf0]		1 System ...	0.000s	0.000s	Scalar				

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

File: multiply.c:245 multiply4

Line	Source

```
230                }
231            }
232        }
233      //}
234    }
235
236    void multiply4(int msize, int tidx, int numt, TYPE a[][NUM], TYPE b[][NUM], TYPE c[][NUM], TYPE t[][NUM])
237    {
238      //loop vectorization with pragma omp simd
239
240      int i,j,k;
241    //  #pragma omp parallel for collapse (2) //num threads(60)
242      for(i=0; i<msize; i++) {
           [loop in multiply4 at multiply.c:242]
             Scalar loop. Not vectorized: inner loop was already vectorized
             No loop transformations applied
243        for(k=0; k<msize; k++) {
           [loop in multiply4 at multiply.c:243]
             Scalar loop
             No loop transformations applied
244          #pragma omp simd
245          for(j=0; j<msize; j++) {
           [loop in multiply4 at multiply.c:245]
             Vectorized AVX; FMA loop processes Float64 data type(s) and includes FMA
             Loop was unrolled by 4
           [loop in multiply4 at multiply.c:245]
             Scalar peeled loop [not executed]
             Loop was unrolled by 4
           [loop in multiply4 at multiply.c:245]
             Vectorized AVX; FMA remainder loop [not executed] processes Float64 data type(s) and includes FMA
             No loop transformations applied
           [loop in multiply4 at multiply.c:245]
             Scalar remainder loop [not executed]
```

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

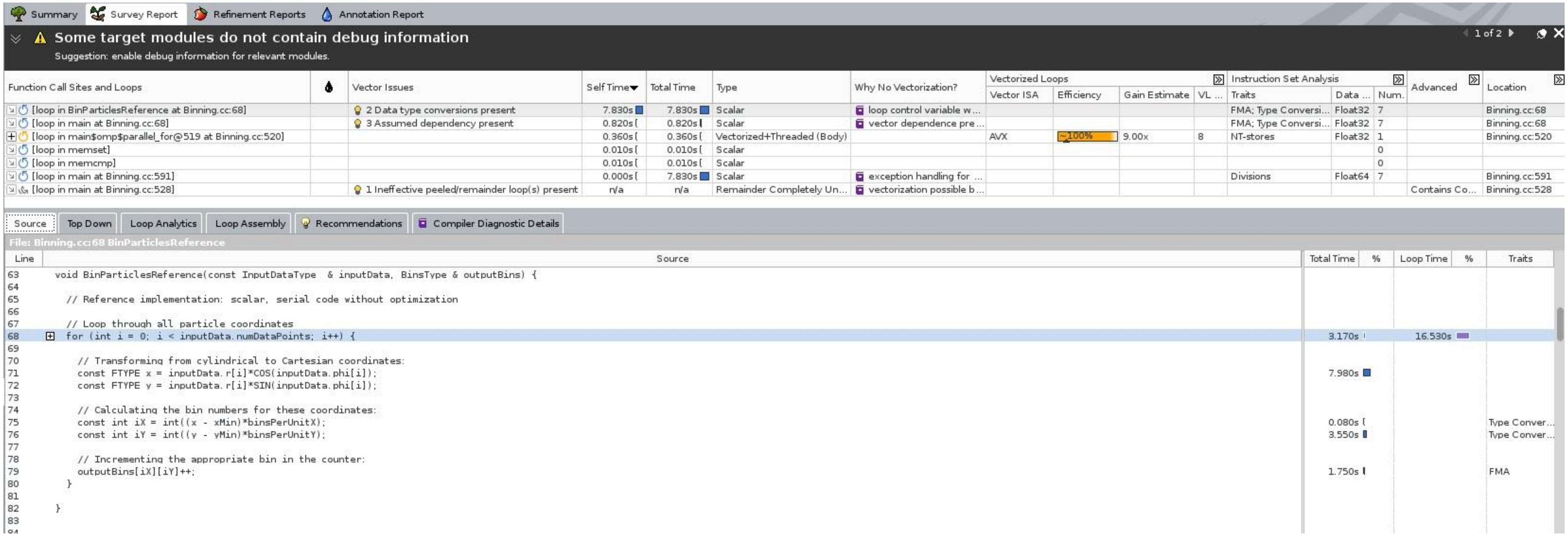
```
for (int i = 0; i < inputData.numDataPoints; i++) {

    // 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				Instruction Set Analysis			Advanced	Location
							Vector ISA	Efficiency	Gain Estimate	VL ...	Traits	Data ...	Num.		
[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				Instruction Set Analysis			Advanced	Location
							Vector ISA	Efficiency	Gain Estimate	VL ...	Traits	Data ...	Num.		
[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				
249						

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)]
===

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 | Tota

```
void
diffusion_openmpv(REAL *restrict f1, REAL *restrict f2, int nx, int ny, int nz,
                  REAL ce, REAL cw, REAL cn, REAL cs, REAL ct,
                  REAL cb, REAL cc, REAL dt, int count) {
    REAL *f1_t = f1;
    REAL *f2_t = f2;

    for (int i = 0; i < count; ++i) {
      for (int z = 0; z < nz; z++) {
        for (int y = 0; y < ny; y++) {
      #pragma simd
          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...	Vectorized Loops: Efficiency	Vectorized Loops: 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..**

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++) { [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	0.140s		4.660s	
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;				

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

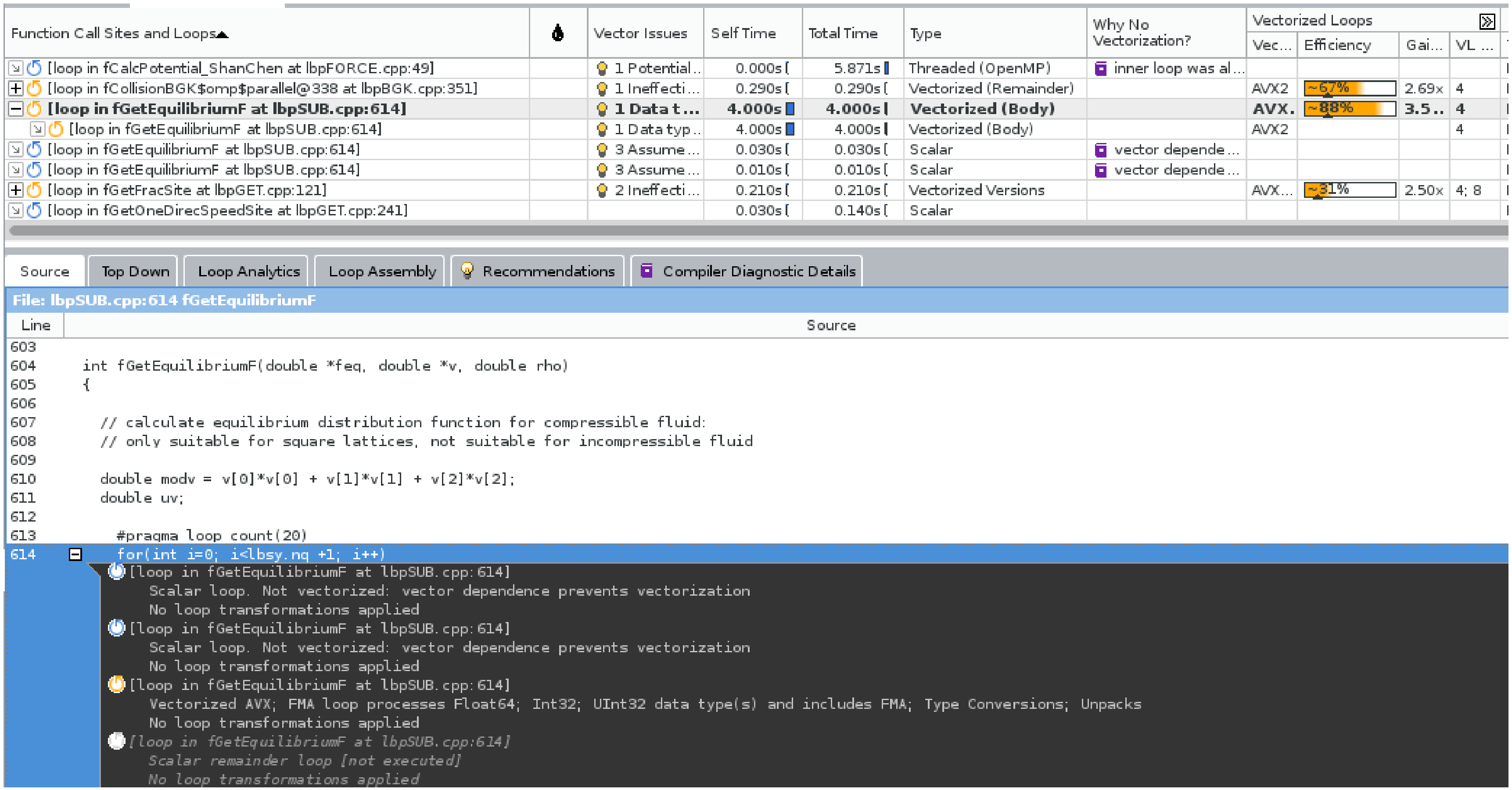
```
int fGetEquilibriumF(double *feq, double *v, double rho)
{

  // calculate equilibrium distribution function for compressible fluid:
  // only suitable for square lattices, not suitable for incompressible fluid

  double modv = v[0]*v[0] + v[1]*v[1] + v[2]*v[2];
  double uv;

    //#pragma loop count(20)
    //for(int i=0; i<lbsy.nq +1; i++)
    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



Questions?

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

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

Source Code

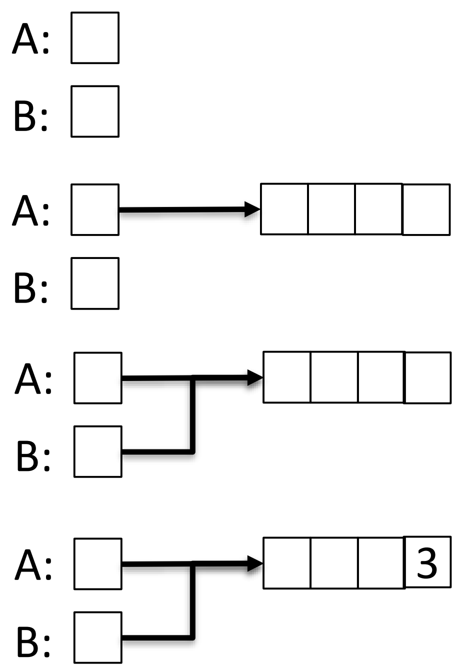
Memory

Double * A;
Double * B;

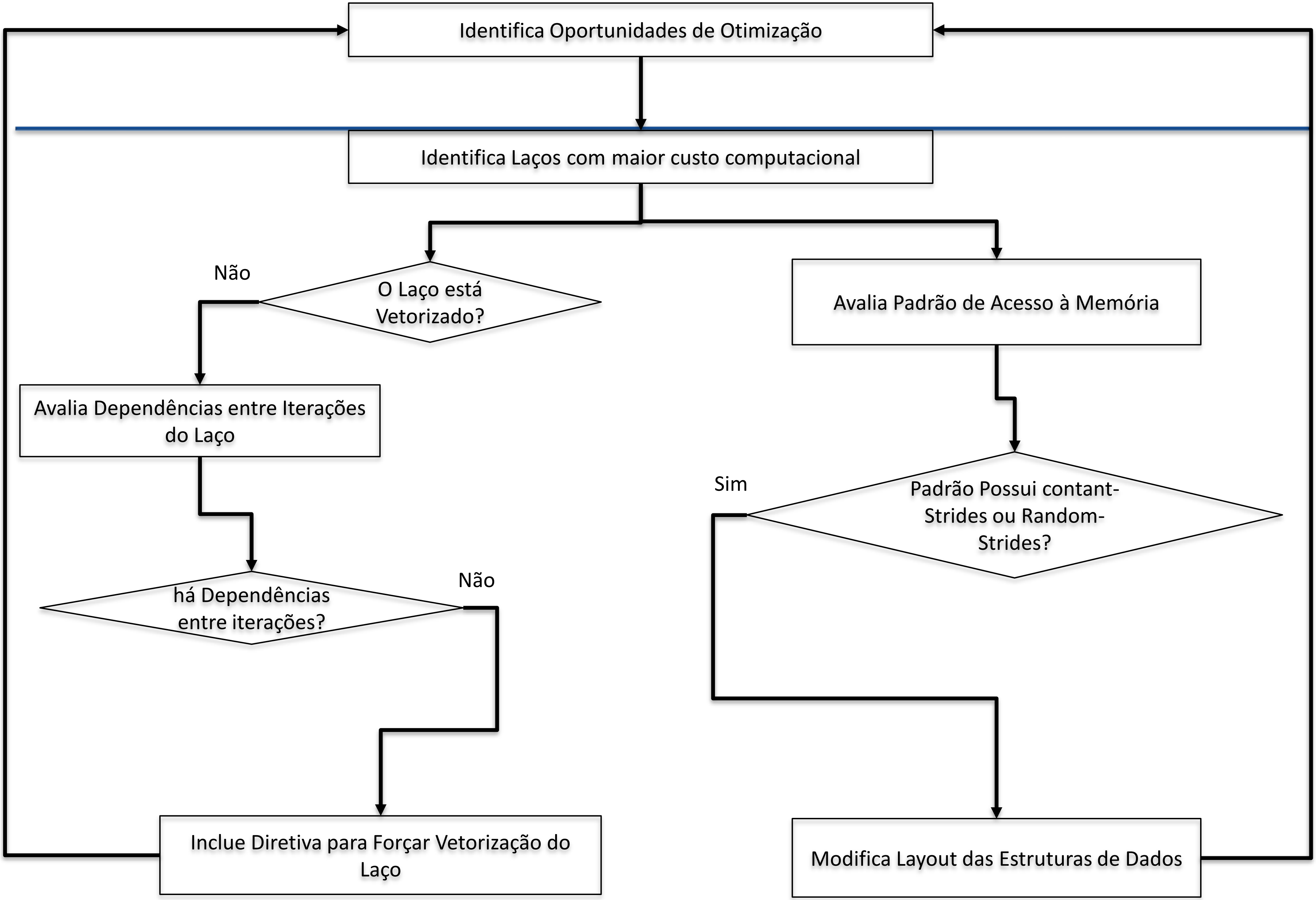
A = (double *)
malloc(4*(sizeof (double)));

B = A;

B[3] = 3;



A and B refers to same array



Identifica Oportunidades de Otimização
Identifica Laços com maior custo computacional
Não
O Laço está Vetorizado?
Avalia Padrão de Acesso à Memória
Avalia Dependências entre Iterações do Laço
Sim
Padrão Possui contant-Strides ou Random-Strides?
há Dependências entre iterações?
Não
Inclue Diretiva para Forçar Vetorização do Laço
Modifica Layout das Estruturas de Dados

Código minicurso	Código git	https://github.com/intel-unesp-mcp/wscad2016-short-course
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		

Hybrid Parallel Architectures

- Heterogeneous computational systems:
  - Scalar and Vector Instructions

Vector Instructions (SIMD)

A7	A6	A5	A4	A3	A2	A1	A0
+							
B7	B6	B5	B4	B3	B2	B1	B0
=							
A7+B7	A6+B6	A5+B5	A4+B4	A3+B3	A2+B2	A1+B1	A0+B0

Scalar Instructions

A
+
B
=
A+B

  - Multi-level memory
    - ❑ RAM Memory;
    - ❑ Multi-level Cache.

Processor 1			Processor 2		
Core 1	Core 2	Core N	Core 1	Core 2	Core N
L1	L1	L1	L1	L1	L1
L2	L2	L2	L2	L2	L2
L3			L3		
RAM					

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)