程序性能优化 Part I

从程序优化角度，我们的课程安排

- Week 1: Tuning for In-core programming
- **Week 2: Tools: Intel compiler and Vtune**
- Week 3: Tuning for Parallel programming
- Week 4: Show case

Intel 编译器介绍

Intel 编译器简介

部分优化参数介绍

编译器优化的方法

Intel 编译器

- 最新版本13.x
- 多语言支持
  - C
  - C++
  - Fortran
  - OpenMP
- GCC 兼容

Intel 编译器

- 多体系结构支持
  - IA-32、Intel® 64（包括AMD Athlon64和Opteron等处理器）、IA-64

  - 跨平台兼容性

Host \ Target	IA-32 Architecture	Intel 64 Architecture	IA-64 Architecture
IA-32 Architecture	Yes	Yes	Yes
Intel 64 Architecture	Yes	Yes	Yes
IA-64 Architecture	No	No	Yes

Intel 编译器

- 多操作系统支持
  - Windows
    - Vista、XP、Server 2003
    - Visual Studio 2008、2005、2003
  - Linux
    - 主流的Linux发布
  - MAC OS

Intel编译器的通用优化参数

Mac*/Linux*	Windows*	描述
-O1	/O1	面向可执行文件大小的优化
-O2	/O2	面向执行速度的优化(默认)
-O3	/O3	O2+进一步的优化：主要面向以浮点运算为主的循环和大规模数据处理的程序
-fast	/fast	集成优化
-g	/Zi	加入调试信息

高级优化：过程间优化 (IPO)

- ip: 源程序文件内部的过程间优化
- ipo: 多个源程序的过程间优化
- 函数内联是ipo中最重要性能优化手段

Mac*/Linux*	Windows*
-ip	/Qip
-ipo	/Qipo

高级优化：指导优化(PGO)

用运行时的信息指导编译器的优化过程

Step 1

Instrumented Compilation

(Mac*/Linux*) icc -prof_gen prog.c
(Windows*) icl -Qprof_gen prog.c

→ Instrumented executable

Step 2

Instrumented Execution

Run program on a typical dataset

→ DYN file containing dynamic info: .dyn

Step 3

Feedback Compilation

(Mac/Linux) icc -prof_use prog.c
(Windows) icl -Qprof_use prog.c

→ Merged DYN summary file: .dpi
Delete old dyn files if you do not want the info included

自动并行化

Mac*/Linux*	Windows*
-parallel	/Qparallel
-par_report[n]	/Qpar_report[n]

- 自动并行化的局限：只能处理简单的情况

OpenMP支持

- 编译指导语句的方式支持线程级并行,支持OpenMP 3.0标准
- 用法
  - OpenMP switches: -openmp : /Qopenmp
  - OpenMP reports: -openmp-report : /Qopenmp-report

```
#pragma omp parallel for
 for (i=0;i<MAX;i++)
     A[i]= c*A[i] + B[i];
```

编译器优化的方法?

- 性能优化的第一步
- 有方向感的try and Test (通用优化->专用优化)
  - 对程序的理解
  - 对编译器输出报告的分析
- 在性能提升的同时留意精度问题

Compiler perspective optimization for Power Dynamic Simulation

Comments	Compiler Switches	Results correct?	Performance improvement	alternative
Intel Compiler 9.0				
Baseline	-O2	Correct	/	
General Opt.	-O3	Correct	5%	
	+ -ipo	Correct	3%	
	+ -prof_gen/use	Correct	9.5%	
Floating point Opt.	+ -ftz -IPF_fltacc -IPF_fma -IPF-fp-relaxed	Correct	19%	
Addressing performance Opt.	+ -fno_alias	WRONG		-restrict (4%)

Floating Point Precision

- First of all
  - Choose the right precision
  - exp() for DP, expf() for SP
- Second
  - Try different options trading-off between precision and performance

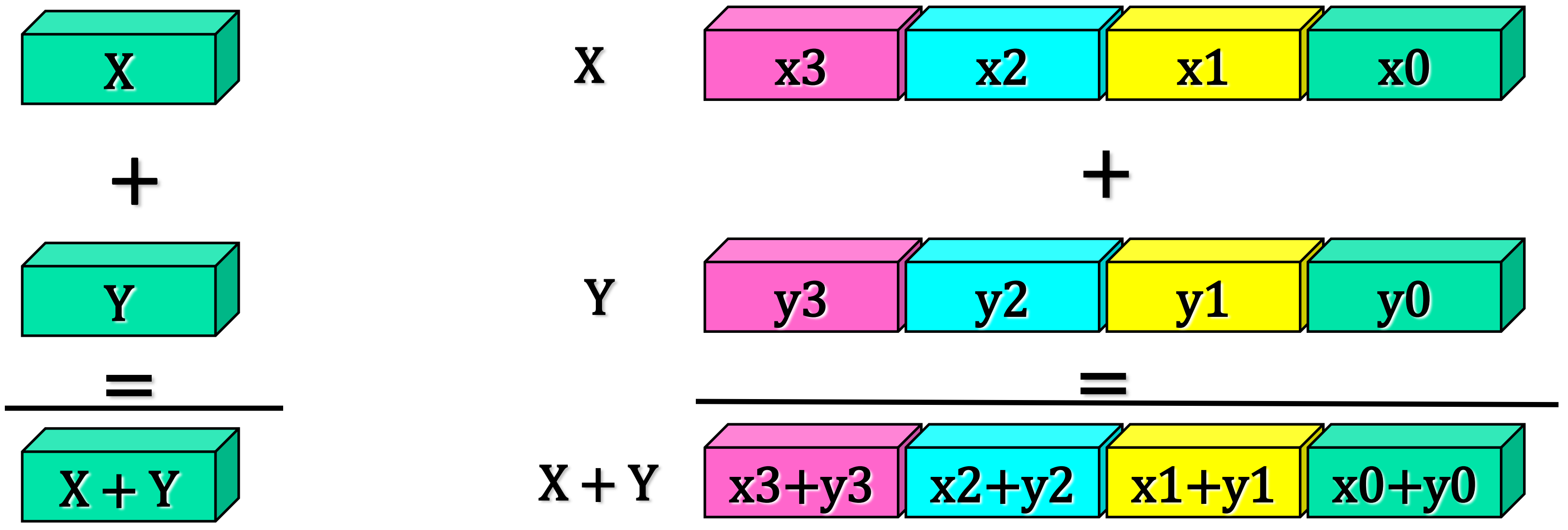
Floating Point Precision

Windows*	Linux*	Description
-Op	-mp	Strict ANSI C and IEEE 754 Floating Point (subset of -Za/-ansi)
-Za	-Xc	Strict ANSI C and IEEE 754
-Qlong_double	-long_double	long double=80, not the default of 64
-Qprec*	-mp1	Precision closer to - but not quite – ANSI ; faster than ANSI
-Qprec_div*	-prec_div* -no-prec-div	Turn off - division into reciprocal multiply
	-prec_sqrt -no-prec-sqrt	Improves precision of square root implementations
-Qpc*n**	-pc*n**	Round to *n* precision. n={32,64,80}
-Qrcd*	-rcd*	Remove code that truncates during float to integer conversions

Vectorization

```
for (i = 0; i < 100; i++)
    xy[i] = x[i] + y[i];
```

- Scalar processing
  - traditional mode
  - one instruction produces one result
- SIMD processing
  - with SSE(2,3,4)
  - one instruction produces multiple results



编译器报告

- OpenMP reports: -openmp-report : /Qopenmp-report

Macintosh*/Linux*	Windows*
-vec_reportn	**-Qvec_reportn**

- Set diagnostic level dumped to stdout

- n=0: No diagnostic information
- n=1: (Default) Loops successfully vectorized
- n=2: Loops not vectorized – and the reason why not
- n=3: Adds dependency Information
- n=4: Reports only non-vectorized loops
- n=5: Reports only non-vectorized loops and adds dependency info

Why Loops Don't Vectorize

- Independence
  - Loop Iterations generally must be independent
- Some relevant qualifiers:
  - Some dependent loops can be vectorized.
  - Most function calls cannot be vectorized.
  - Some conditional branches prevent vectorization.
  - Loops must be countable.
  - Outer loop of nest cannot be vectorized.
  - Mixed data types cannot be vectorized.

Why Loops Don't Vectorize

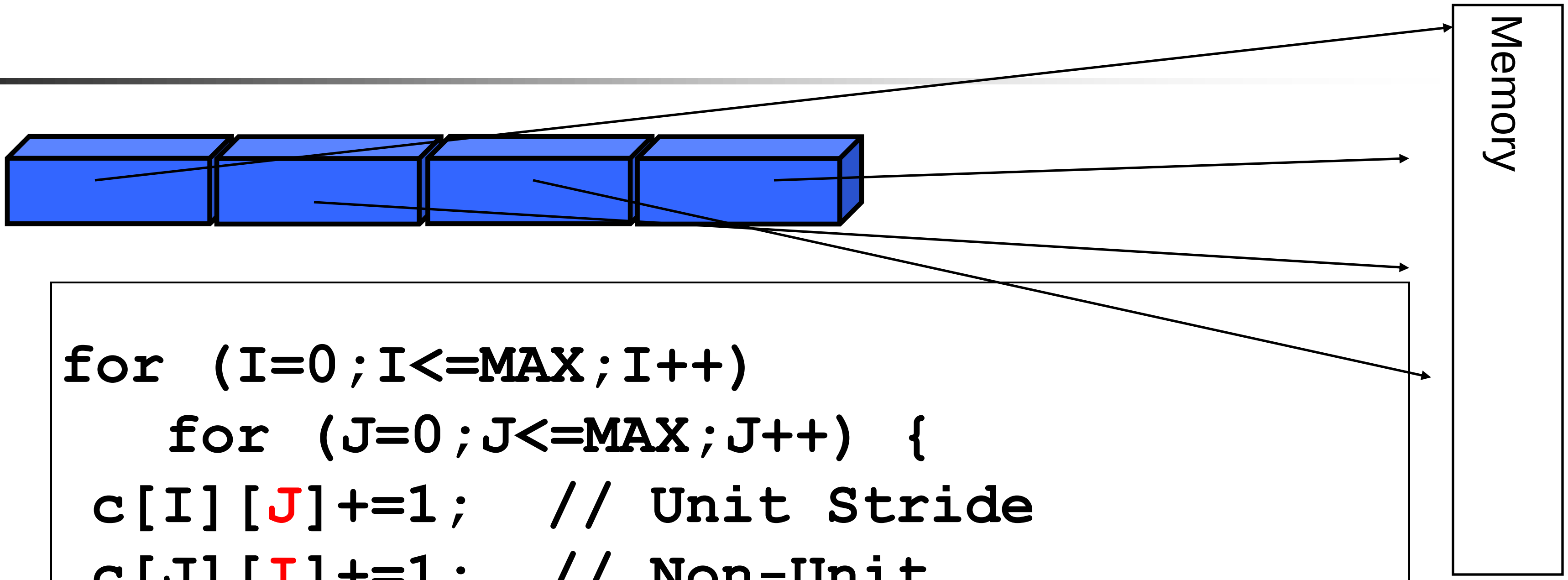
- "Existence of vector dependence"
- "Nonunit stride used"
- "Mixed Data Types"
- "Unsupported Loop Structure"
- "Contains unvectorizable statement at line XX"
- There are more reasons loops don't vectorize but we will disucss the reasons above

"Existence of Vector dependency"

- Usually, indicates a real dependency between iterations of the loop, as shown here:

```
for (i = 0; i < 100; i++)
    x[i] = A * x[i + 1];
```

“Nonunit stride used”



```
for (I=0;I<=MAX;I++)
   for (J=0;J<=MAX;J++) {
 c[I][J]+=1;  // Unit Stride
 c[J][I]+=1;  // Non-Unit
 A[J*J]+=1;   // Non-unit
 A[B[J]]+=1;  // Non-Unit
 if (A[MAX-J])=1 last1=J;}// Non-Unit
```

End Result: Loading Vector may take more cycles than executing operation sequentially.

“Mixed Data Types”

An example:

```
int howmany_close(double *x, double *y)
{ int withinborder=0;
  double dist;
  for(int i=0;i<MAX;i++) {
    dist=sqrtf(x[i]*x[i] + y[i]*y[i]);
    if (dist<5) withinborder++;
  }
}
```

- Mixed data types are possible – but complicate things
  - i.e.: 2 doubles vs 4 ints per SIMD register
- Some operations with specific data types won’t work

"Unsupported Loop Structure"

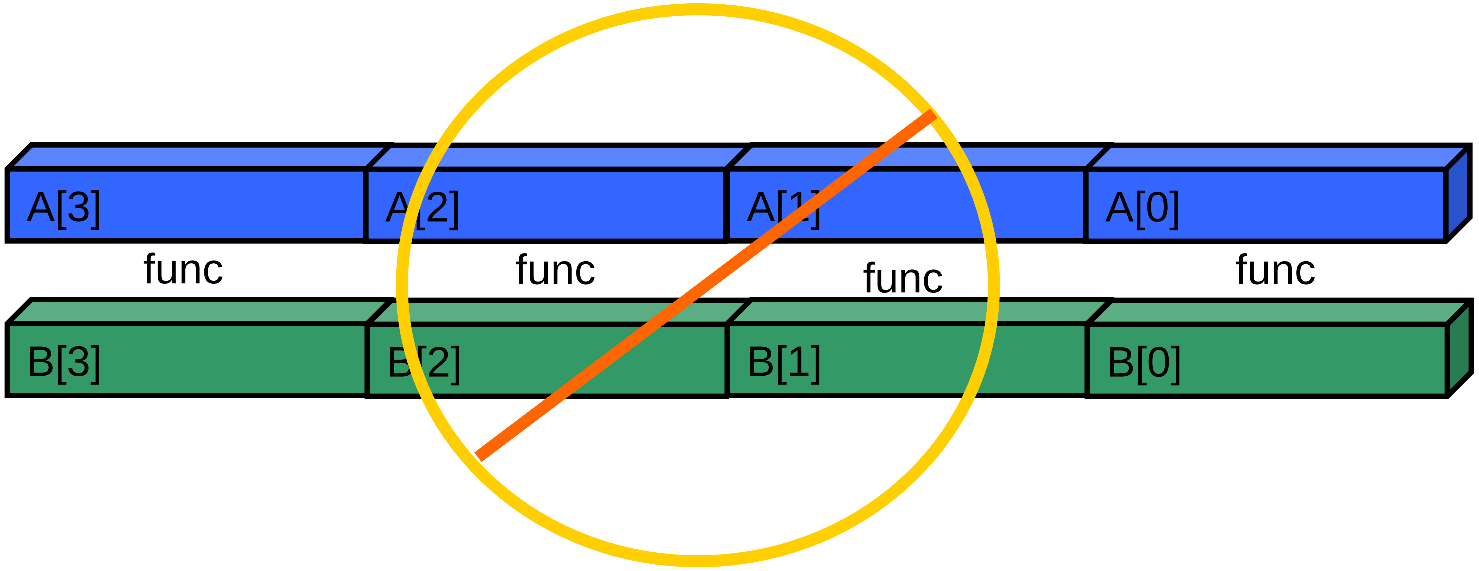
- Example:

```
struct _xx  {
  int data;
  int bound; } ;
doit1(int *a, struct _xx *x)  {
for (int i=0; i<x->bound; i++) a[i] = 0;
```

- An unsupported loop structure means the loop is not countable, or the compiler for whatever reason can't construct a run-time expression for the trip count.

"Contains unvectorizable statement"

```
for (i=1;i<nx;i++) {
   B[i] = func(A[i]); }
```



提供编译器向量化的帮助信息

编译指示（pragma)	语法(Semantics)
`#pragma ivdep`	编译器忽略不能确定的数据依赖。
`#pragma vector always [assert]`	循环能够向量化时，编译器忽略量化开销分析，总进行向量化。 循环不能向量化时， 编译器通过"assert" 语句提供错误信息。
`#pragma novector`	提示编译器，即使在符合向量化条件下，也不进行循环向量化. 有时，向量化可能会带来性能开销， 我们无需编译器进行向量化。
`#pragma vector aligned / unaligned`	向量化时， 提示编译器使用对齐内存的操作指令（或不对齐内存的操作指令）。
`#pragma vector temporal / nontemporal`	在IA32与Intel 64的架构上， 指示编译器使用 temporal/non-temporal 内存存取指令。 多个变量时，使用逗号进行分隔。

使用simd 指示字强制循环向量化
– #pragma simd

- 语法：#pragma simd [<clause-list>]
  - 强制一个循环向量化的语法
  - 程序员：需判断一个循环是否应该被向量化
  - 编译器：强制将循环向量化或者给出错误信息

字句（**Clause**）	语义（**Semantics**）
none	强制将最内层循环进行向量化， 忽略数据间的依赖性。
vectorlength *(n_1[, n_2]…)*	指示向量化时， 向量长度(2， 4， 8， 16)。 编译器编译时，从中选择。
private *(var_1, var_2, …, var_N)*	每次循环的私有标量。 多个循环同时执行时，各循环获得相同初始值。 变量的最终结果为最后执行循环的该变量值。
linear *(var_1:$step_1$, …, var_N:$step_N$)*	声明变量的步长。 步长值为正数，且为向量长度的整数倍。
reduction *(operator:var_1, var_2,…, var_N)*	声明reduction 变量。 变量在循环结束时， 通过相应的 reduction 操作累计最后的结果。
*[*no*]*assert	向量化失败时，编译器通过assert 语句给出(或不给出)提示。缺省为编译器给出向量化失败的提示。

Alignment

- Allocating aligned memory
  - Static memory
    - Allocated by compiler/linker
    - Add `__attribute__((aligned(n)))` in front variable declaration
    - Applies to global/local static variables as well as stack/auto variables
  - Dynamic memory
    - Allocated by language runtime
    - Use `__mm_aligned_malloc(size, alignment_bytes)`
    - Pair it with `__mm_aligned_free()`
  - Using Intel® Threading Building Blocks
    - Dynamic memory allocation API that supports the Intel® MIC Architecture
    - Use `scalable_aligned_malloc()/scalable_aligned_free()`
    - Include `<tbb/scalable_allocator.h>`, and link with `–ltbbmalloc`
  - In Fortran:

    !DEC$ ATTRIBUTES ALIGN: 64 :: var

Profiling

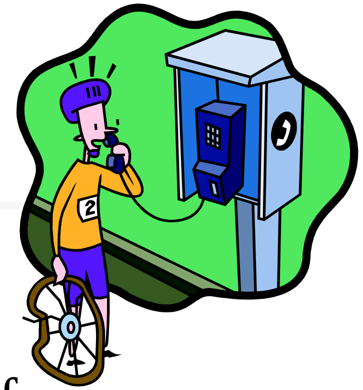
Profiling

Profiling means collecting data about how a program executes.

The two major kinds of profiling are:

- Subroutine profiling
- Hardware counters

Two Ways to Track Location



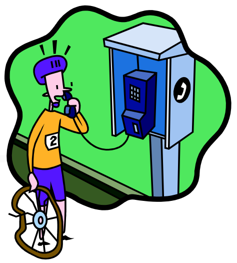
- Problem: I need to know where you spend MOST of your time.
- Statistical Solution: **I call you on your cellular phone every 30 minutes and ask you to report your location; then I plot a location histogram.**
- Instrumentation Solution: **I install a special phone booth at the entrance of every site you plan to visit. As you enter any site, you first go into the booth, call the operator to get the exact time, and then call me and tell me where you are and when you got there. Then I plot your route of travel on a map.**

The Statistical Solution: I Call You!

- ADVANTAGES:
  - Low Intrusion (Overhead): Reciting you location and then hanging up is not much of an interrupt.
  - No Construction Costs or Site Requirements
  - Catches you wherever you go with great precision.
- DISADVANTAGES:

Statistical Significance: There are some places I might not catch you, if you don’t go there often and/or you don’t stay very long, but I have to ask myself if I really care.

The Instrumentation Solution: You call me when you get there!



- ADVANTAGES
  - I know where you were immediately
  - I know how long you were at each place
  - I know how many times you visited each place
- DISADVANTAGES
  - I don’t know about details of the places you go
  - Granularity is course
  - You waste a lot of time
  - I have to build all those damn phone booths which expands the space in each site you visit.

Subroutine Profiling

Subroutine profiling means finding out how much time is spent in each routine.

The 90-10 Rule: Typically, a program spends 90% of its runtime in 10% of the code.

Subroutine profiling tells you what parts of the program to spend time optimizing and what parts you can ignore.

Specifically, at regular intervals (e.g., every millisecond), the program takes note of what instruction it's currently on.

Profiling Example

On Intel compilers systems:

`icc -O -g -p` …

The **`-g -p`** options tell the compiler to set the executable up to collect profiling information.

Running the executable generates a file named **`gmon.out`**, which contains the profiling information.

While for gcc, you need

`gcc -O -g -pg` …

Profiling Example (cont'd)

When the run has completed, a file named **`gmon.out`** has been generated.

Then:

```
gprof executable
```

produces a list of all of the routines and how much time was spent in each.

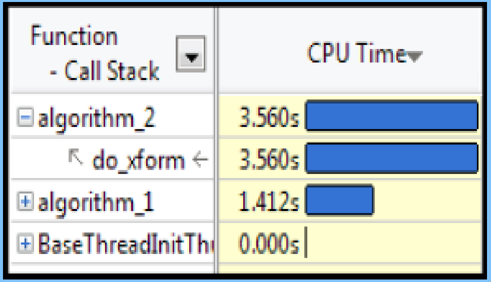
Profiling Result

```
 %   cumulative   self              self     total
time   seconds   seconds    calls  ms/call  ms/call  name
27.6     52.72    52.72   480000     0.11     0.11  longwave_ [5]
24.3     99.06    46.35      897    51.67    51.67  mpdata3_ [8]
 7.9    114.19    15.13      300    50.43    50.43  turb_ [9]
 7.2    127.94    13.75      299    45.98    45.98  turb_scalar_ [10]
 4.7    136.91     8.96      300    29.88    29.88  advect2_z_ [12]
 4.1    144.79     7.88      300    26.27    31.52  cloud_ [11]
 3.9    152.22     7.43      300    24.77   212.36  radiation_ [3]
 2.3    156.65     4.43      897     4.94    56.61  smlr_ [7]
 2.2    160.77     4.12      300    13.73    24.39  tke_full_ [13]
 1.7    163.97     3.20      300    10.66    10.66  shear_prod_ [15]
 1.5    166.79     2.82      300     9.40     9.40  rhs_ [16]
 1.4    169.53     2.74      300     9.13     9.13  advect2_xy_ [17]
 1.3    172.00     2.47      300     8.23    15.33  poisson_ [14]
 1.2    174.27     2.27   480000     0.00     0.12  long_wave_ [4]
 1.0    176.13     1.86      299     6.22   177.45  advect_scalar_ [6]
 0.9    177.94     1.81      300     6.04     6.04  buoy_ [19]
...
```

Intel® VTune™ Amplifier XE
Performance Profiler

Where is my application…

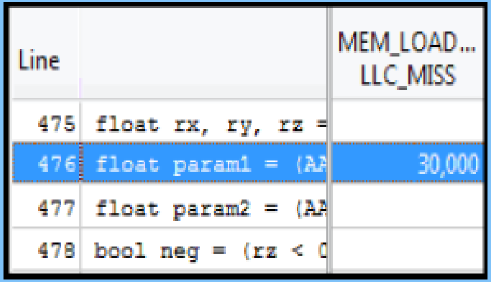
Spending Time?



Function - Call Stack	CPU Time
⊟algorithm_2	3.560s
↖ do_xform ←	3.560s
⊞algorithm_1	1.412s
⊞BaseThreadInitThı	0.000s

- Focus tuning on functions taking time
- See call stacks
- See time on source

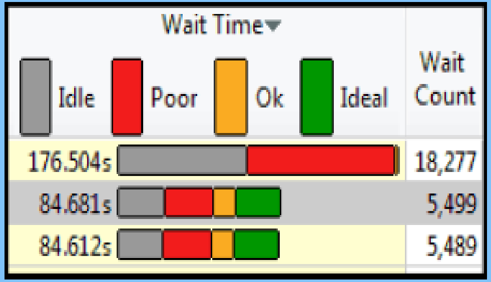
Wasting Time?



Line		MEM_LOAD... LLC_MISS
475	float rx, ry, rz =	
476	float param1 = (AA	30,000
477	float param2 = (AA	
478	bool neg = (rz < 0	

- See cache misses on your source
- See functions sorted by # of cache misses

Waiting Too Long?



Wait Time (Idle / Poor / Ok / Ideal)	Wait Count
176.504s	18,277
84.681s	5,499
84.612s	5,489

- See locks by wait time
- Red/Green for CPU utilization during wait

- Windows & Linux
- Low overhead
- No special recompiles

✓ 热点分析
✓ 并行度分析
✓ 锁和等待分析
✓ 对比分析

We improved the performance of the latest run 3 fold. We wouldn't have found the problem without something like Intel® VTune™ Amplifier XE.

Claire Cates
Principal Developer, SAS Institute Inc.

- Please login166.111.68.173 with putty
  - user:test
  - passwd:test@tsinghua
- Create your dir in home dir
  - mkdir xuew
- Copy cases to your dir
  - 向量化 /opt/intel/composer_xe_2013.2.146/Samples/en_US/C/vec_samples
  - /opt/intel/vtune_amplifier_xe/samples/en/C/tachyon_vtune_amp_xe.tgz
- Tutorial Doc:
  - 向量化 http://software.intel.com/sites/products/documentation/doclib/stdxe/2013/composerxe/tutorials/lin/cmp_vec_c/index.htm
  - Vtune (网络学堂hotspots_amplxe_lin)