allinea

Leaders in parallel software development tools

Optimize code for Intel Xeon Phi

Discovering bottlenecks without pain

JDEV2013

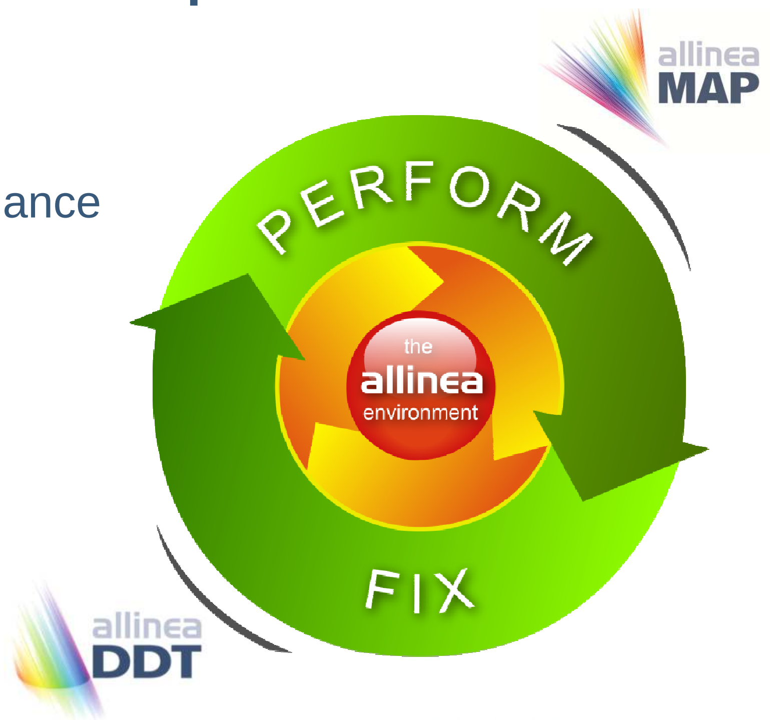
www.allinea.com

Agenda

- Introduction
- Allinea MAP and optimizing for Xeon Phi
- Conclusion

Allinea Unified environment

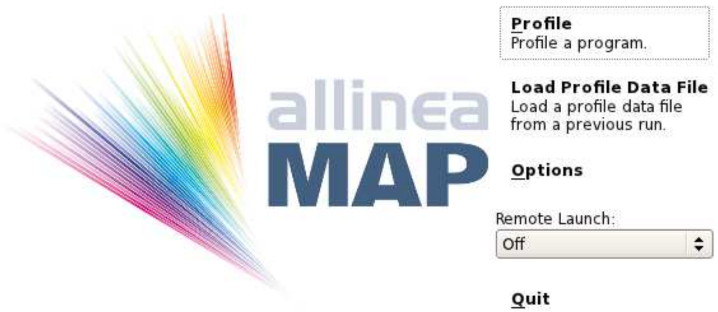
- **A modern integrated environment for HPC developers**
- **Supporting the lifecycle of application development and improvement**
  - **Allinea DDT :** Productively debug code
  - **Allinea MAP :** Enhance application performance
- **Designed for productivity**
  - Consistent easy to use tools
  - Enables effective HPC development
- **Improve system usage**
  - Fewer failed jobs
  - Higher application performance

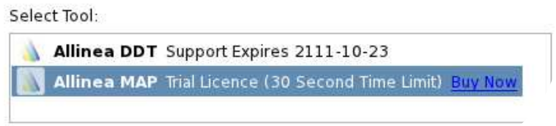


allinea
www.allinea.com

Allinea MAP
Increase application performance

- **Parallel profiler designed for:**
  - C/C++, Fortran
  - Multiprocess code
    - Interdependent or independent processes
  - Multithreaded code
    - Monitor the main threads for each process
  - **Accelerated codes**
    - GPUs, **Intel Xeon Phi**





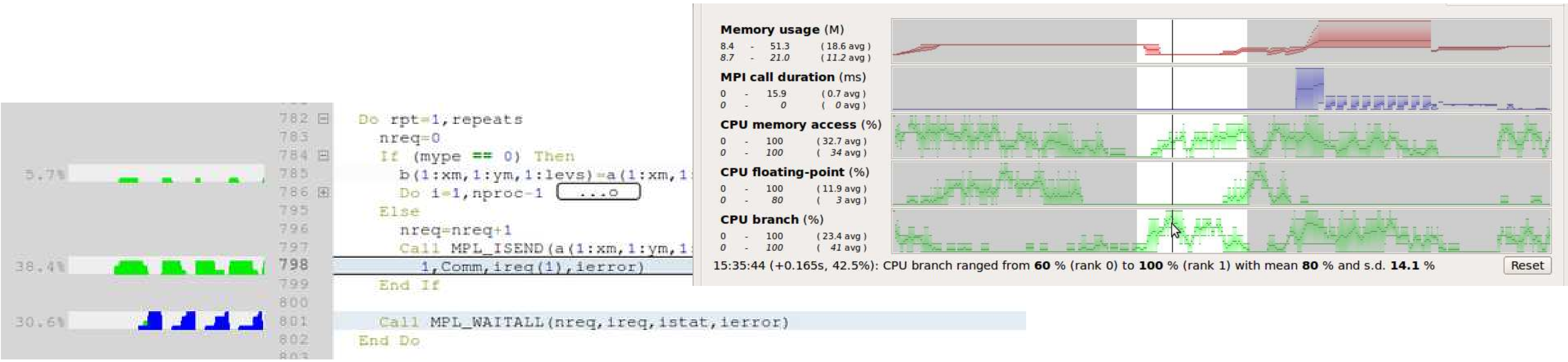
- **Improve productivity :**
  - Helps you detect performance issues quickly and easily
  - Tells you immediately where your time is spent in your source code
  - Helps you to optimize your application efficiently

allinea
www.allinea.com

Allinea MAP
Find performance issues quickly

- **Look at the entire application on real data sets**
  - Visualize the entire run at full scale, not just reduced sets
  - Zoom in to explore iterations, functions and loops

- **Understand the nature of bottlenecks**
  - Source code viewer pinpoints bottleneck locations
  - CPU, MPI, I/Os and memory metrics identify the cause



Allinea MAP
The missing link in HPC tools

- **Unique profiling methodology (even on Intel Xeon Phi)**
  - No need to instrument your code
  - Just 5% wall-clock overhead
  - Small output files (10-20Mb is typical)

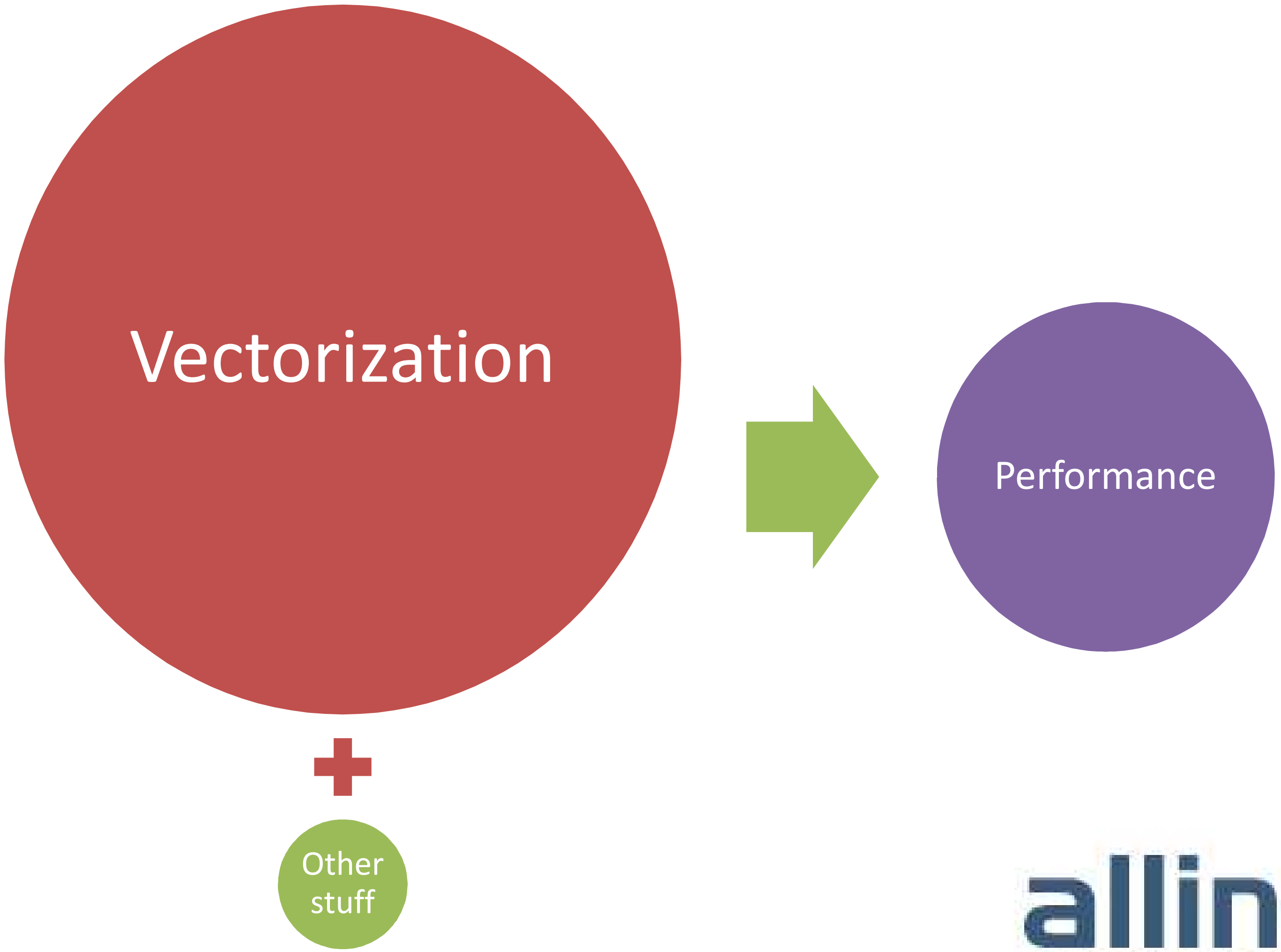
- **Benefit:**
  - 80% of the profiling activity can now be done by end-users
  - Helps to focus on the right bit of the application
  - Brings more value to the expertise of application support

Optimizing for the Xeon Phi
Where do you start?

"Code that's well-optimized for the host usually performs pretty well on the cards"

- Pretty much everyone

Optimizing for the Xeon Phi
But what matters?



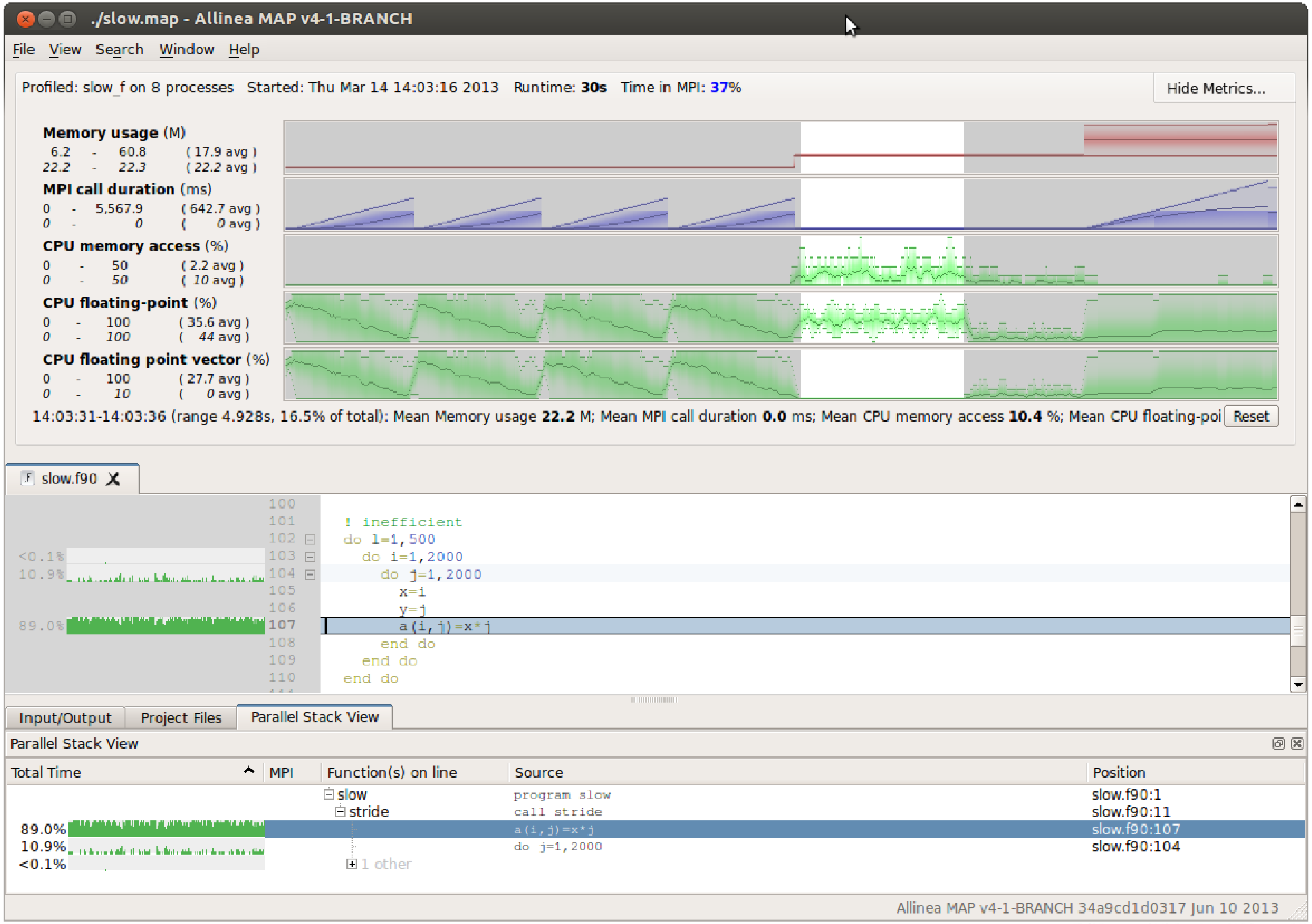
allinea
www.allinea.com

Optimizing for the Xeon Phi
Is my code well-vectorized?

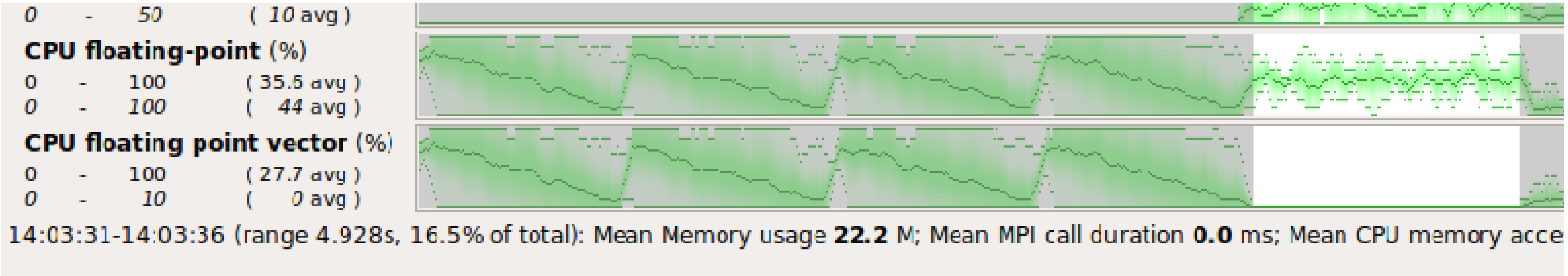
```
mg.f(2432): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(2431): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(993): (col. 13) remark: LOOP WAS VECTORIZED.
mg.f(992): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(991): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(243): (col. 7) remark: loop was not vectorized: existence of vector depende
nce.
mg.f(993): (col. 13) remark: LOOP WAS VECTORIZED.
mg.f(992): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(991): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(753): (col. 13) remark: loop was not vectorized: vectorization possible but
 seems inefficient.
mg.f(762): (col. 13) remark: loop was not vectorized: vectorization possible but
 seems inefficient.
mg.f(749): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(746): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(993): (col. 13) remark: LOOP WAS VECTORIZED.
mg.f(992): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(991): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(2255): (col. 16) remark: loop was not vectorized: existence of vector depen
dence.
mg.f(2254): (col. 13) remark: loop was not vectorized: not inner loop.
mg.f(2251): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(2433): (col. 13) remark: LOOP WAS VECTORIZED.
mg.f(2433): (col. 13) remark: loop was not vectorized: not inner loop.
mg.f(2432): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(2431): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(2433): (col. 13) remark: LOOP WAS VECTORIZED.
mg.f(2433): (col. 13) remark: loop was not vectorized: not inner loop.
mg.f(2432): (col. 10) remark: loop was not vectorized: not inner loop.
mg.f(2431): (col. 7) remark: loop was not vectorized: not inner loop.
mg.f(527): (col. 7) remark: loop was not vectorized: nonstandard loop is not a v
ectorization candidate.
mg.f(552): (col. 7) remark: loop was not vectorized: nonstandard loop is not a v
ectorization candidate.
mg.f(1150): (col. 7) remark: loop was not vectorized: loop was transformed to me
mset or memcpy.
mg.f(1150): (col. 7) remark: loop was not vectorized: loop was transformed to me
mset or memcpy.
mg.f(1645): (col. 7) remark: loop was not vectorized: loop was transformed to me
mset or memcpy.
mg.f(1655): (col. 7) remark: loop was not vectorized: loop was transformed to me
```

… maybe?

Optimizing for the Xeon Phi
Is my code well-vectorized?

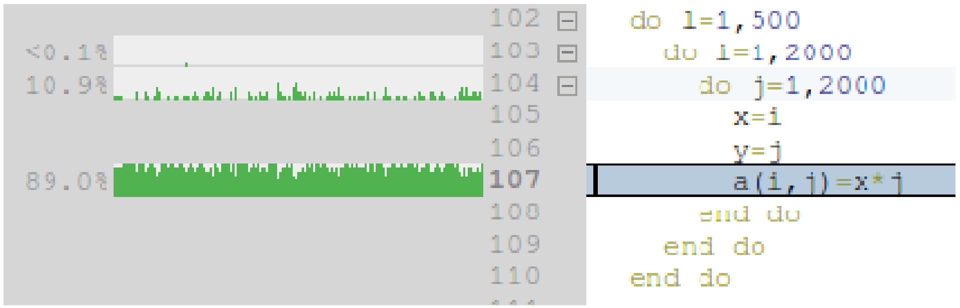


Optimizing for the Xeon Phi
Is my code well-vectorized?

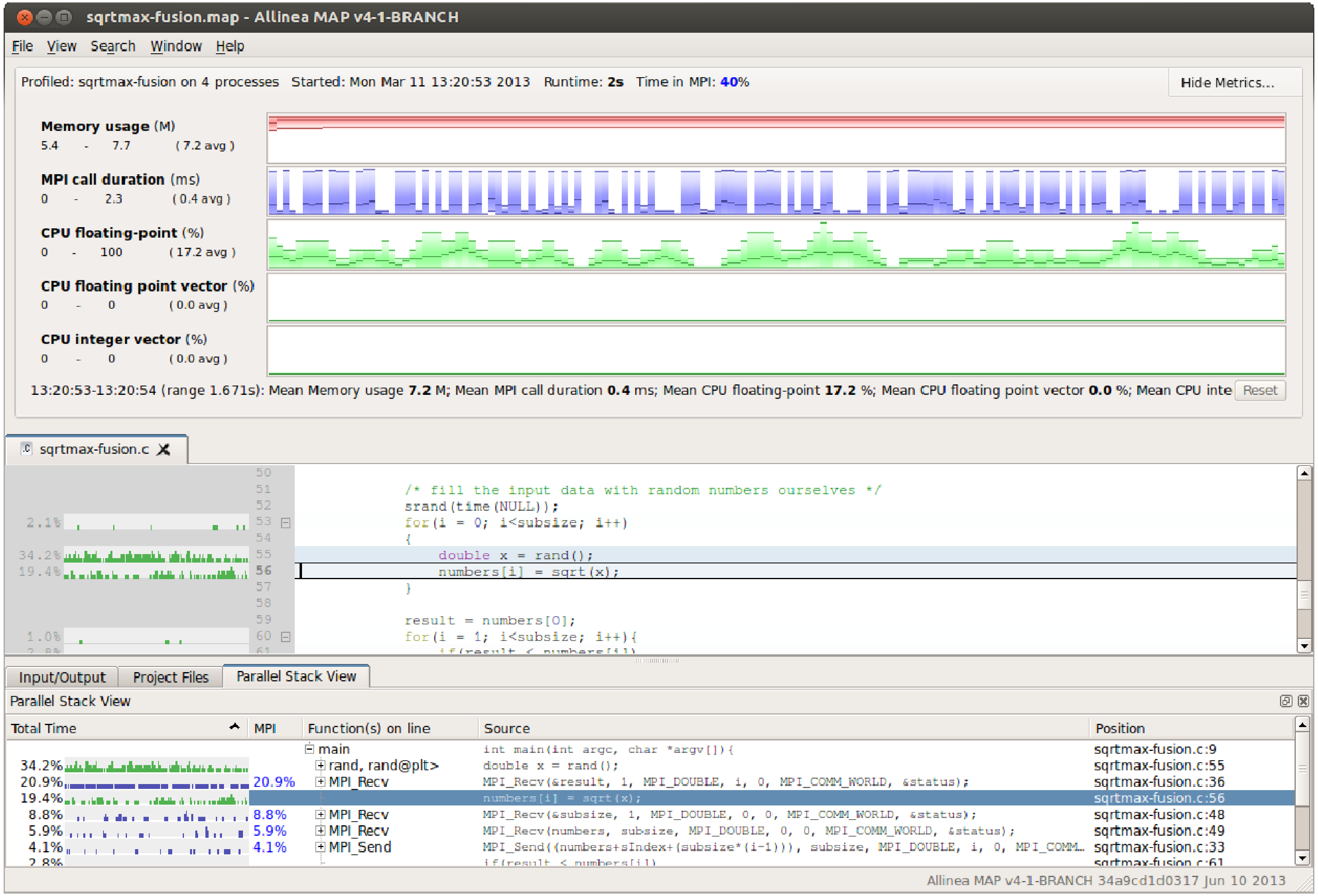


Not in this loop

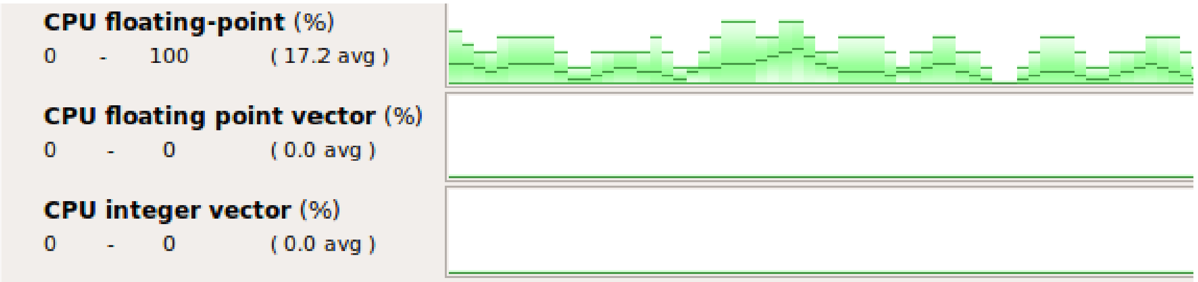
(16.5% of total time)



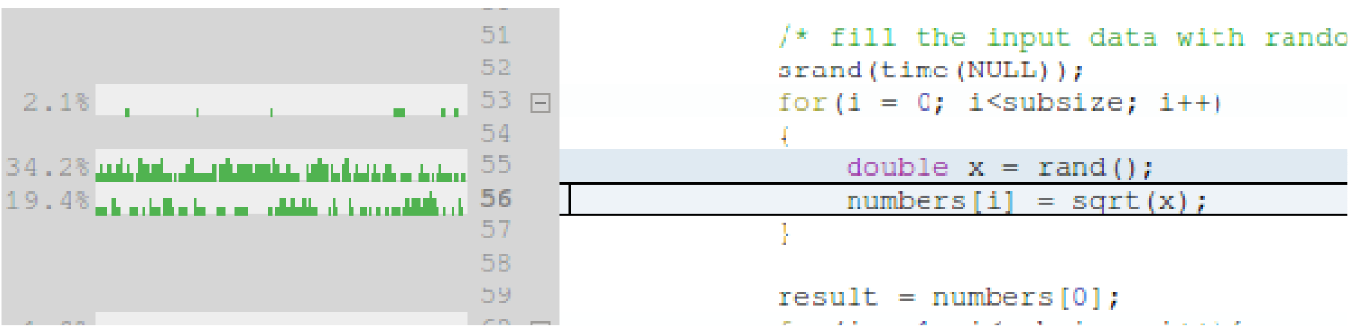
Optimizing for the Xeon Phi
Non-obvious tradeoffs



Optimizing for the Xeon Phi
Non-obvious tradeoffs

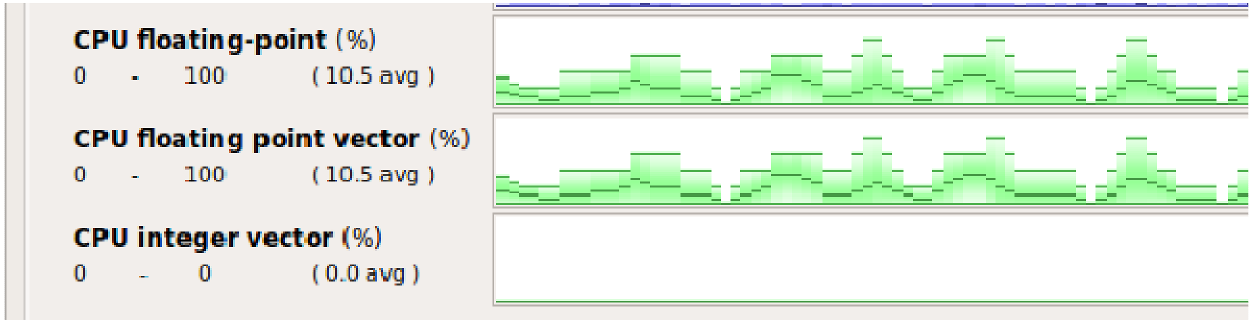


Here a loop taking 55% of total runtime isn't vectorized at all

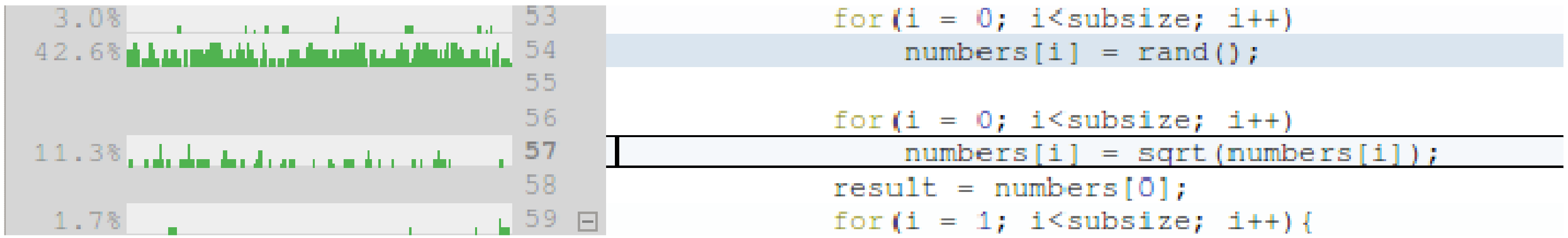


Taking the unvectorizable rand() out of the loop allows the sqrt workload to be fully-vectorized – reverse loop fusion!

Optimizing for the Xeon Phi
Non-obvious tradeoffs



Now the floating-point workload is fully-vectorized



But all the time is being spent in the random number generation, so that’s what really needs to be optimized

Optimizing for the Xeon Phi
Know your tools

Random Number Function Vectorization

Submitted by Ronald W Green ... on Fri, 09/07/2012 - 16:31

Categories: Intel® Many Integrated Core Architecture , Vectorization , Intel® C++ Compiler , Intel® Fortran Compiler , C/C++ , Fortran , Developers , Linux* , Advanced

Tags: Random Number Function Vectorization

Drand48 Vectorization in C/C++ Goodman, Steve9700.00000000000
Compiler Methodology for Intel® MIC Architecture

Vectorization Essentials, Random Number Function Vectorization

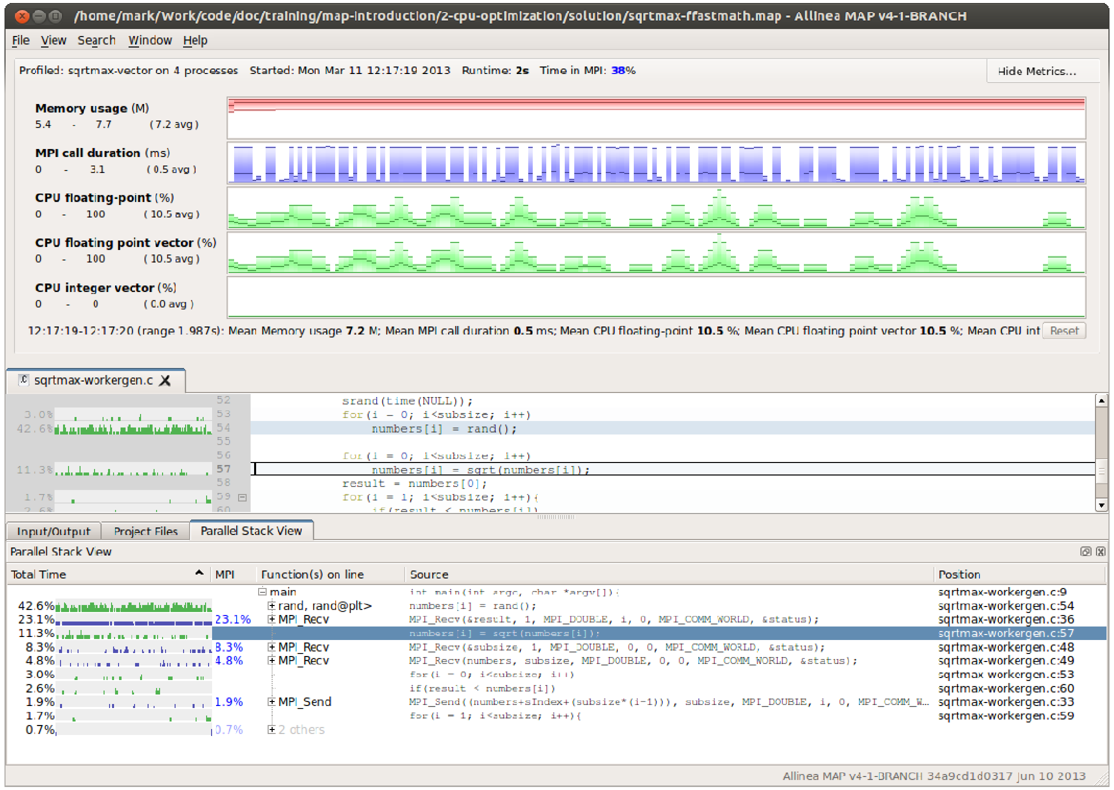
The Intel 13.0 Product Compiler now supports random number auto- vectorization of the drand48 family of random number functions in C/C++ and RANF and Random_Number functions in Fortran. Vectorization is supported through the Intel Short Vector Math Library (SVML).

Supported C/C++ Functions:

```
double drand48(void);
double erand48(unsigned short xsubi[3]);
long int lrand48(void);
long int nrand48(unsigned short xsubi[3]);
```

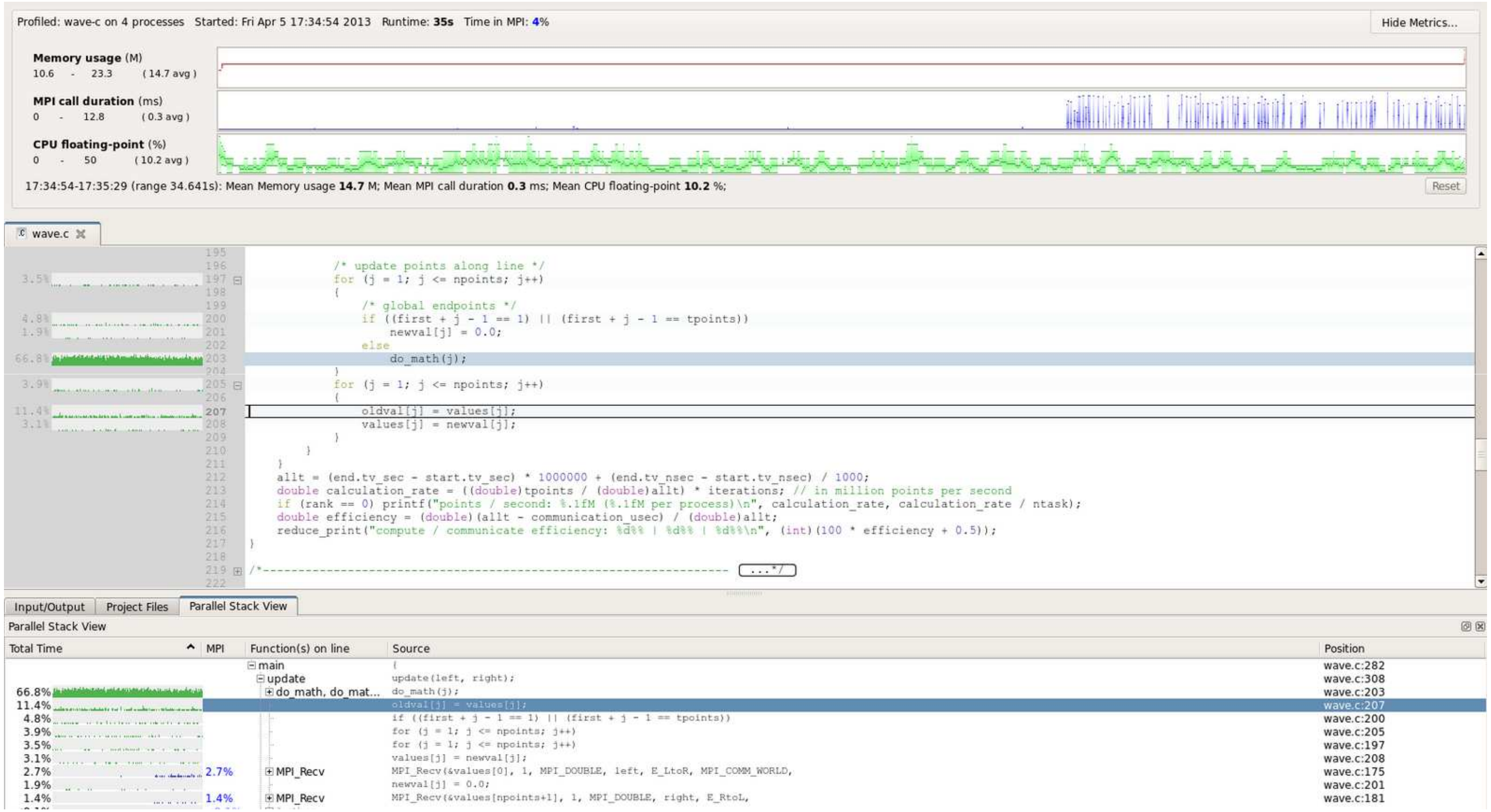
Replace rand() with Intel's vectorized version and re-fuse the loop to retain temporal cache locality benefits

Optimizing for the Xeon Phi
The full picture



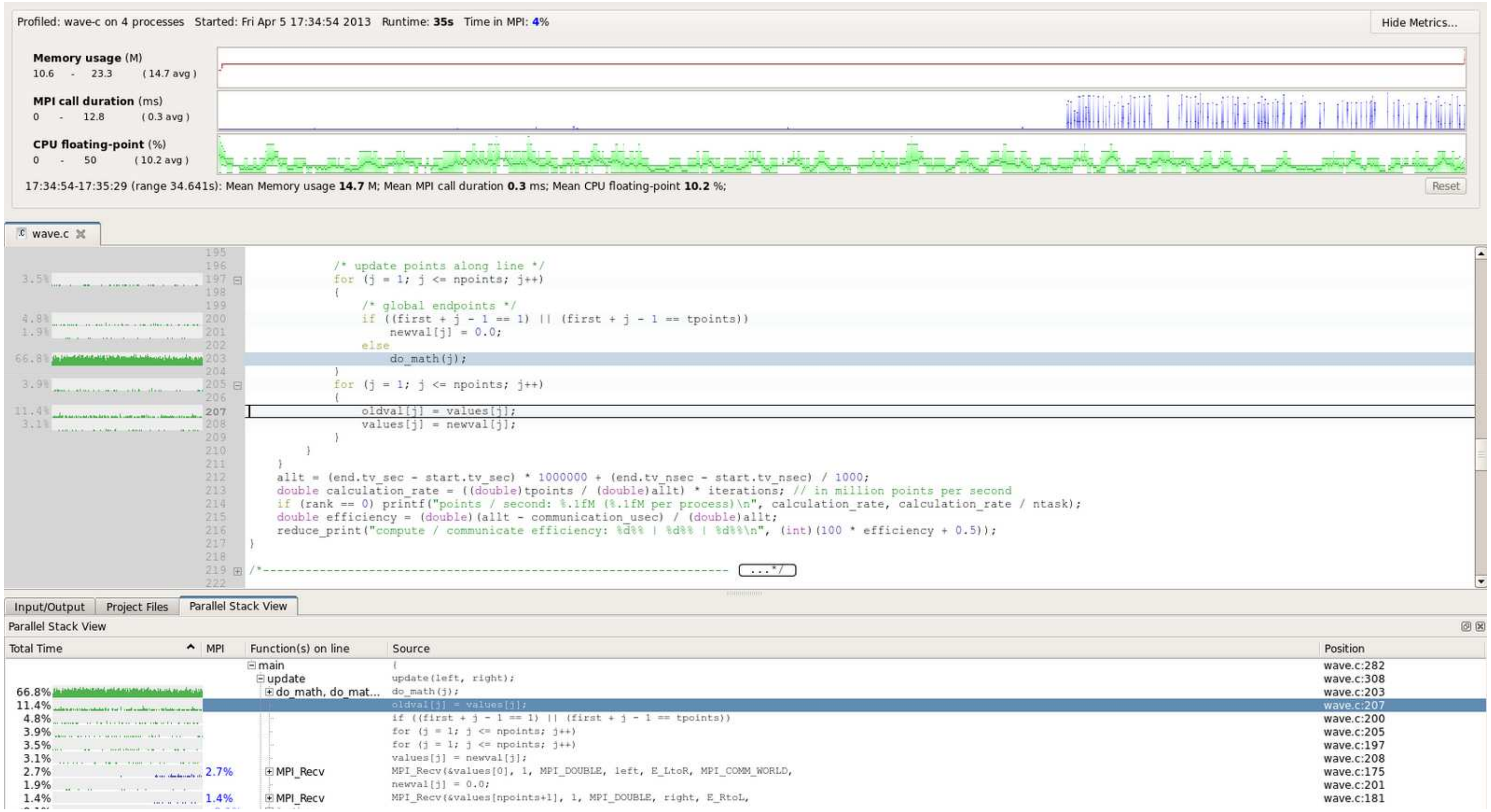
You need to see the full picture to spot these tradeoffs – Allinea MAP shows you the way

Optimizing for the Xeon Phi
Running on the card



Allinea MAP runs *with full metrics* on Xeon Phi cards!

Optimizing for the Xeon Phi
Running on the card



This makes it easy to compare and learn versus the host

Summary

- **Allinea tools are the premier Xeon Phi development environment**
  - See at a glance which loops to vectorize
  - Full profiling metrics available on the Xeon Phi cards
  - Unified interface with Allinea DDT keeps you productive, whatever you're working on

allinea

Leaders in parallel software development tools

TO LEARN MORE ABOUT ALLINEA PRODUCTS

Attend Allinea technical tutorial & presentation !

Technical tutorial :	Room PA2 “Gregory”	Friday 11h00-12h30
Presentation :	Amphi “Becquerel”	Friday 13h15-13h45

Thank you

Your contacts :

- Technical Support team : support@allinea.com
- Sales team : sales@allinea.com

www.allinea.com