allinea

Leaders in parallel software development tools

Paving the Road Ahead for Software Development in HPC

David Lecomber
Allinea Software
david@allinea.com

Allinea Software

- **Our mission: to make HPC software development fast, simple and successful**
  - **A modern integrated environment for HPC developers**
  - **Scalable tools for any scale of system**

- **Supporting the lifecycle of application development and improvement**
  - **Allinea DDT :** Productively debug code
  - **Allinea MAP :** Enhance application performance

- **Designed for productivity**
  - Consistent integrated easy to use tools
  - Enables effective use of HPC resources and expertise

www.allinea.com

Major Supercomputing Centers

Oak Ridge National Laboratory

cea

Blue Waters
Sustained Petascale Computing

NERSC
National Energy Research Scientific Computing Center

HLRS

Argonne National Laboratory

NSF

TACC

Berkeley Lab

SHARCNET™

ersity of California
Lawrence Livermore National Laboratory

SNIC

Technische Universität Dresden

Los Alamos National Laboratory
Est. 1943

Jülich Forschungszentrum

GENCI

ONERA
The French Aerospace Lab

BSC
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

ICHEC
Irish Centre for High-End Computing

CINECA

lrz

Science & Technology Facilities Council

epcc

hpc
CYMRU WALES

NCHC
財團法人國家實驗研究院
國家高速網路與計算中心
National Center for High-performance Computing

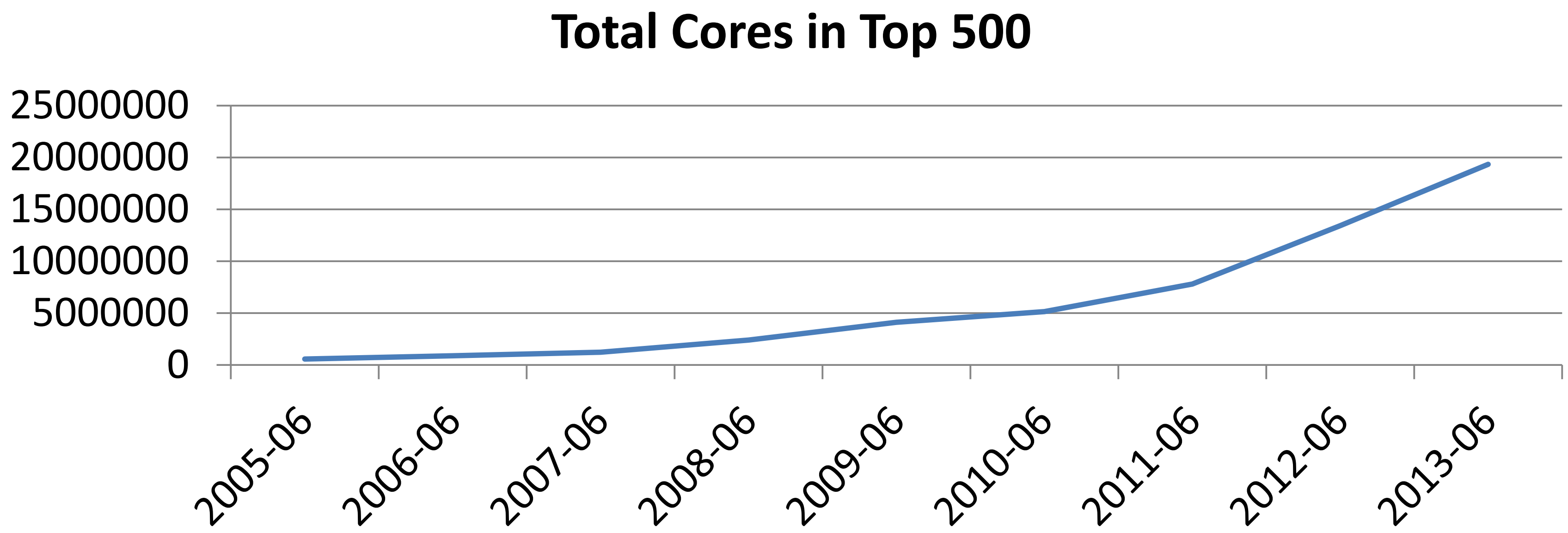
King Abdulaziz University

IFERC

Saudi Aramco

allinea
www.allinea.com

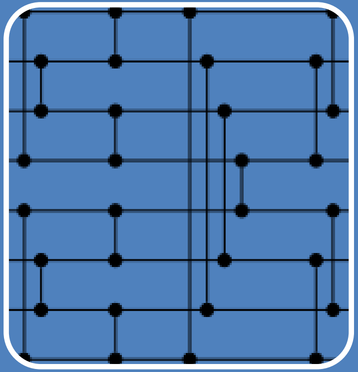
Inexorable March of Scale



- How do we define "HPC" today?
  - Top 500 place now requires ~6,000 cores
  - Coprocessors and accelerators - 15-20% of real HPC machines
- "Build it and they will come"?

allinea www.allinea.com

Some Software Challenges for the Extreme



Algorithmic: Compilers are not enough!

- Restructure for SIMD threads and vectorization
- Fundamental changes: Do we really need FFTs here?
- Rediscover PRAM and 0-1 Sorting Networks(!)

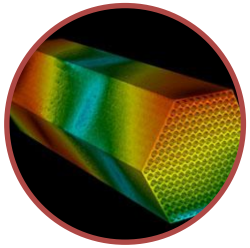
Programmer Efficiency

- MPI alone is not sufficient: Hybrid required
- Performance trade-offs harder to understand
- Software bugs harder to fix

Tackling Software Challenges

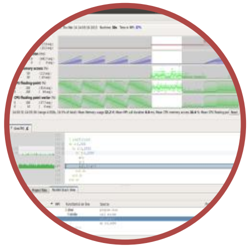
CRESTA

Collaborative Research into Exascale Systemware,Tools and Applications



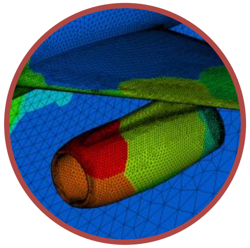
Applications

- Biomolecular systems
- Fusion energy
- Weather prediction
- Engineering



Software Environment

- Debugging
- Profiling
- Auto-tuning



Systemware

- Numerical libraries
- Pre/Postprocessing
- In-situ Visualization
- Heterogeneous programming

Three Challenges for tools

Scalability

- Speed and Simplification

Heterogeneity

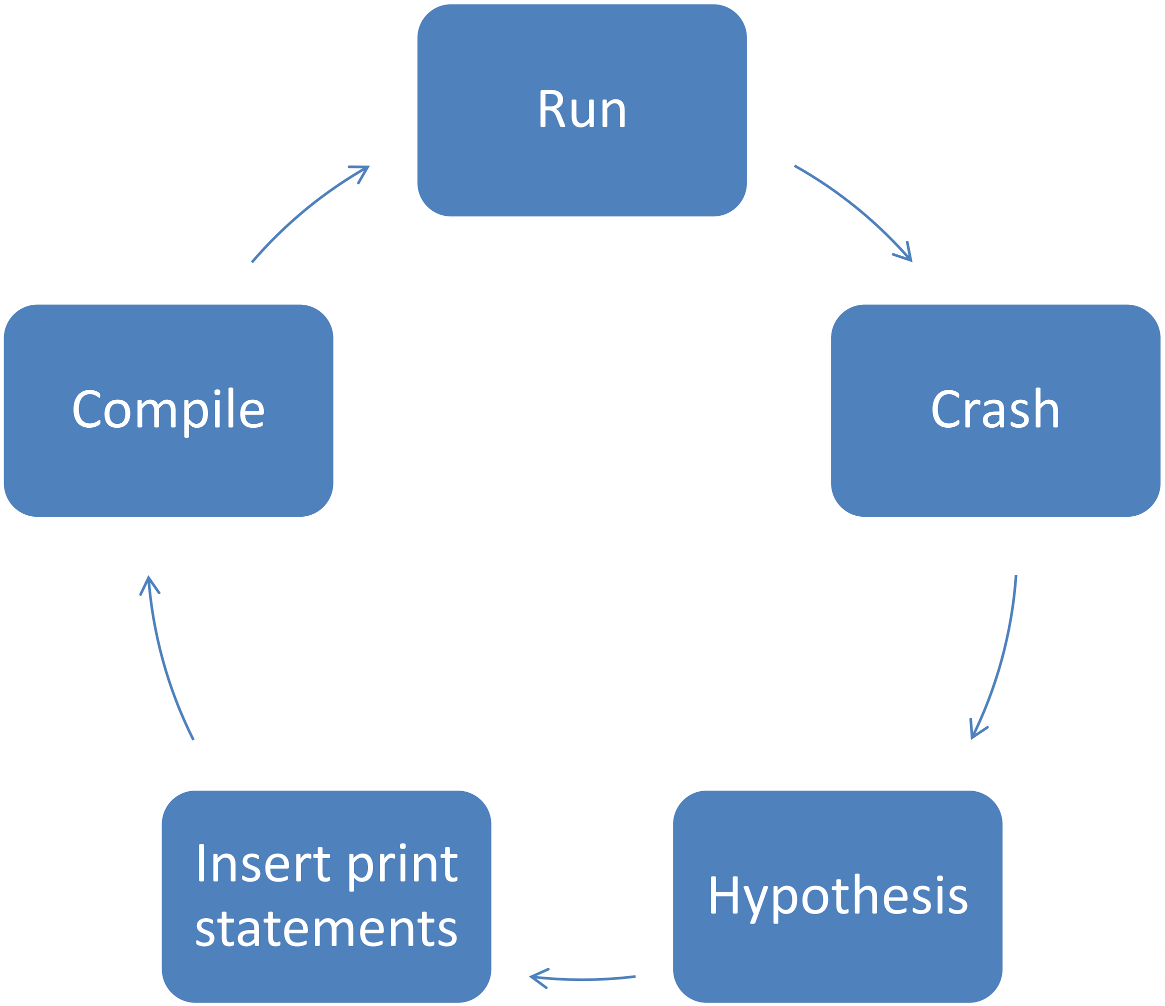
- Accelerators and Coprocessors

Adoption

- Ease of Use and Education

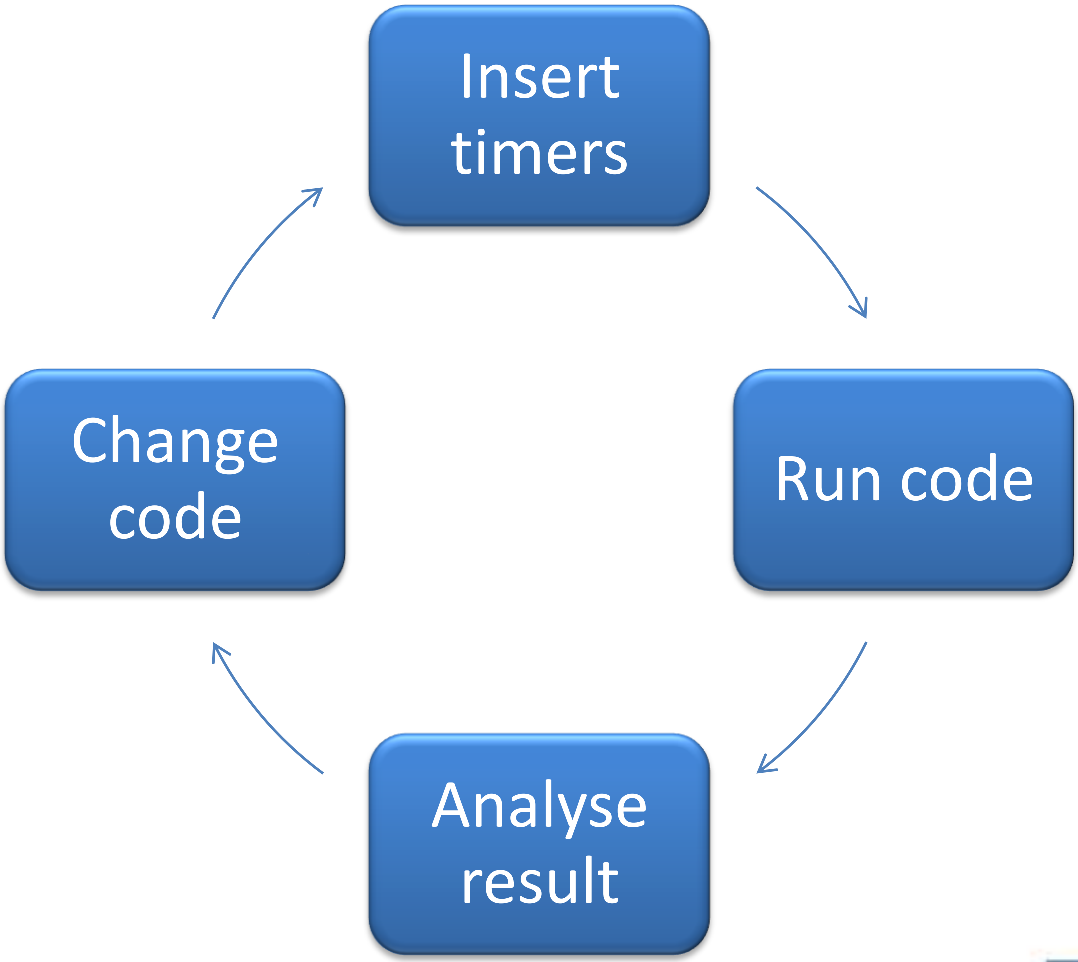
www.allinea.com

Debugging in practice…



www.allinea.com

Optimization in practice…



Exploding Parallelism

Titan

- 18,688 nodes
- 18,688 NVIDIA Kepler K20 GPUs
- 299,008 CPU cores
- 50,233,344 CUDA cores

Tianhe-2

- 16,000 nodes
- 48,000 Intel Xeon Phi
- 32,000 Ivy Bridge
- 3,120,000 cores
- 11,328,000 hardware threads

Do the workflows “work”?

Allinea DDT
Fix software problems, fast

- **Powerful graphical debugger designed for :**
  - C/C++, Fortran, UPC, …
  - MPI, OpenMP and mixed-mode code
  - Accelerators and coprocessors

- **Unified interface with Allinea MAP :**
  - One interface eliminates learning curve
  - Spend more time on your results

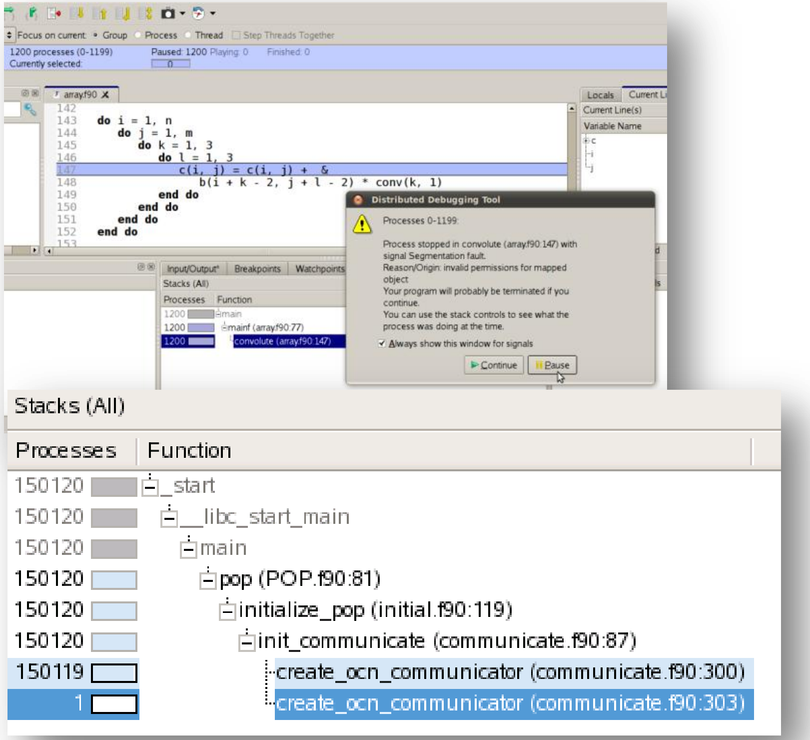
- **Slash your time to develop :**
  - Reproduces and triggers your bugs instantly
  - Helps you easily understand where issues come from quickly
  - Helps you to fix them as swiftly as possible

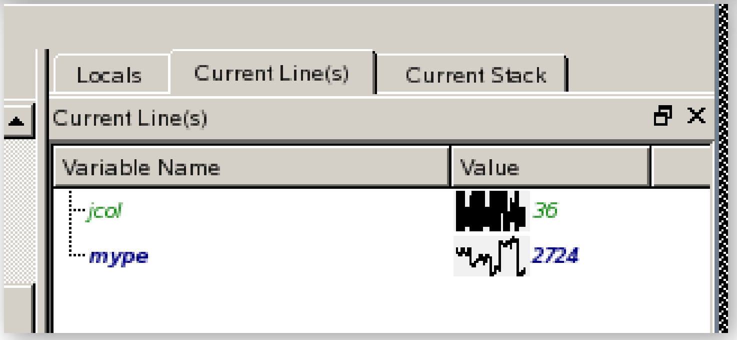
allinea
www.allinea.com

Allinea DDT: Scalable debugging by design

- ***Where* did it happen?**
  - Allinea DDT leaps to source automatically
  - Merges stacks from processes and threads
- ***How* did it happen?**
  - Some faults evident instantly from source
- ***Why* did it happen?**
  - Real-time data comparison and consolidation
  - Unique “Smart Highlighting” – colouring differences and changes
  - Sparklines comparing data across processes

– ***Force* crashes to happen?**
  - Memory debugging makes many random bugs appear every time



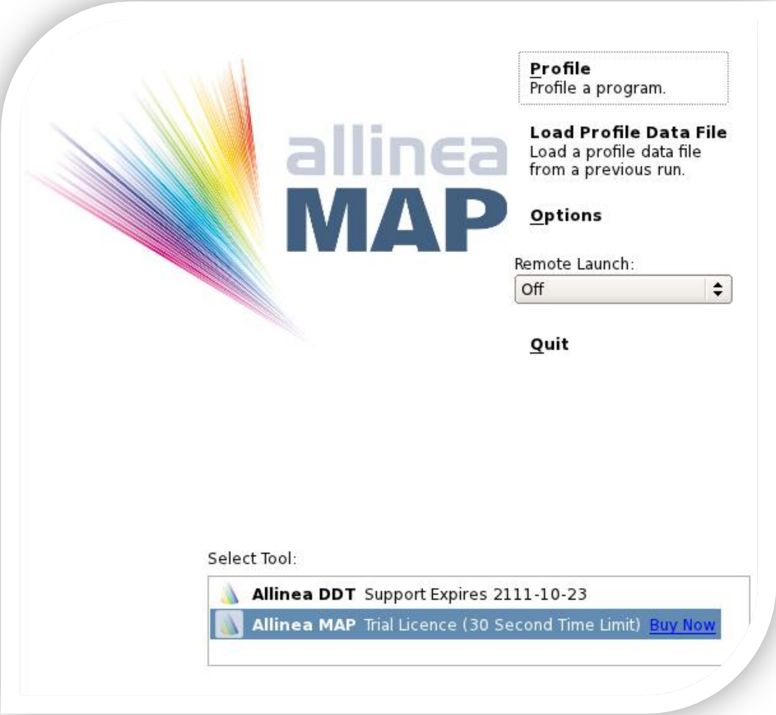


Example

- HPC code fails on 98,304 cores
- Random processes crashing
- Printf? Which processes and where?
- Too costly to repeat
- Allinea DDT finds cause first time

Allinea MAP
Increase application performance

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



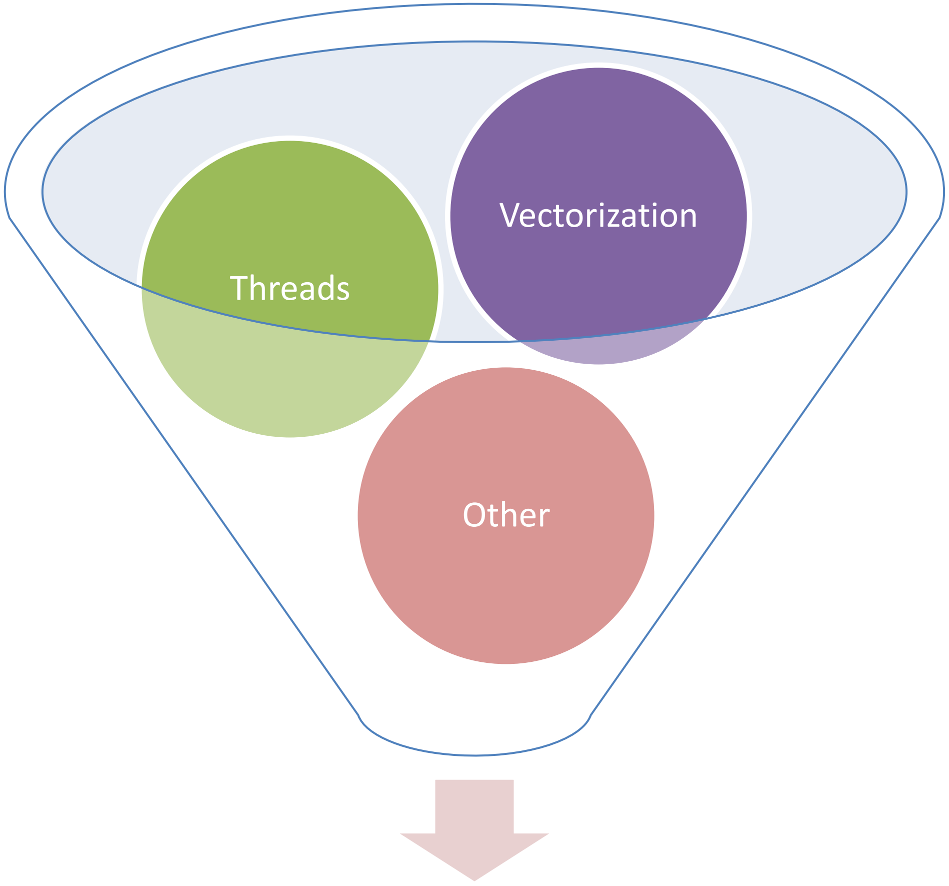
- **Improve productivity :**
  - Helps you detect performance issues quickly and easily
  - Tells you immediately where your time is spent in your source code
  - Helps you to optimize your application efficiently

allinea
www.allinea.com

Simplicity and Capability

- Click and run profiling for HPC!
- <5% runtime overhead
- 20Mb output files
- No instrumentation needed
- Run regularly – or in regression tests

Optimizing for the Xeon Phi
But what matters?



Performance

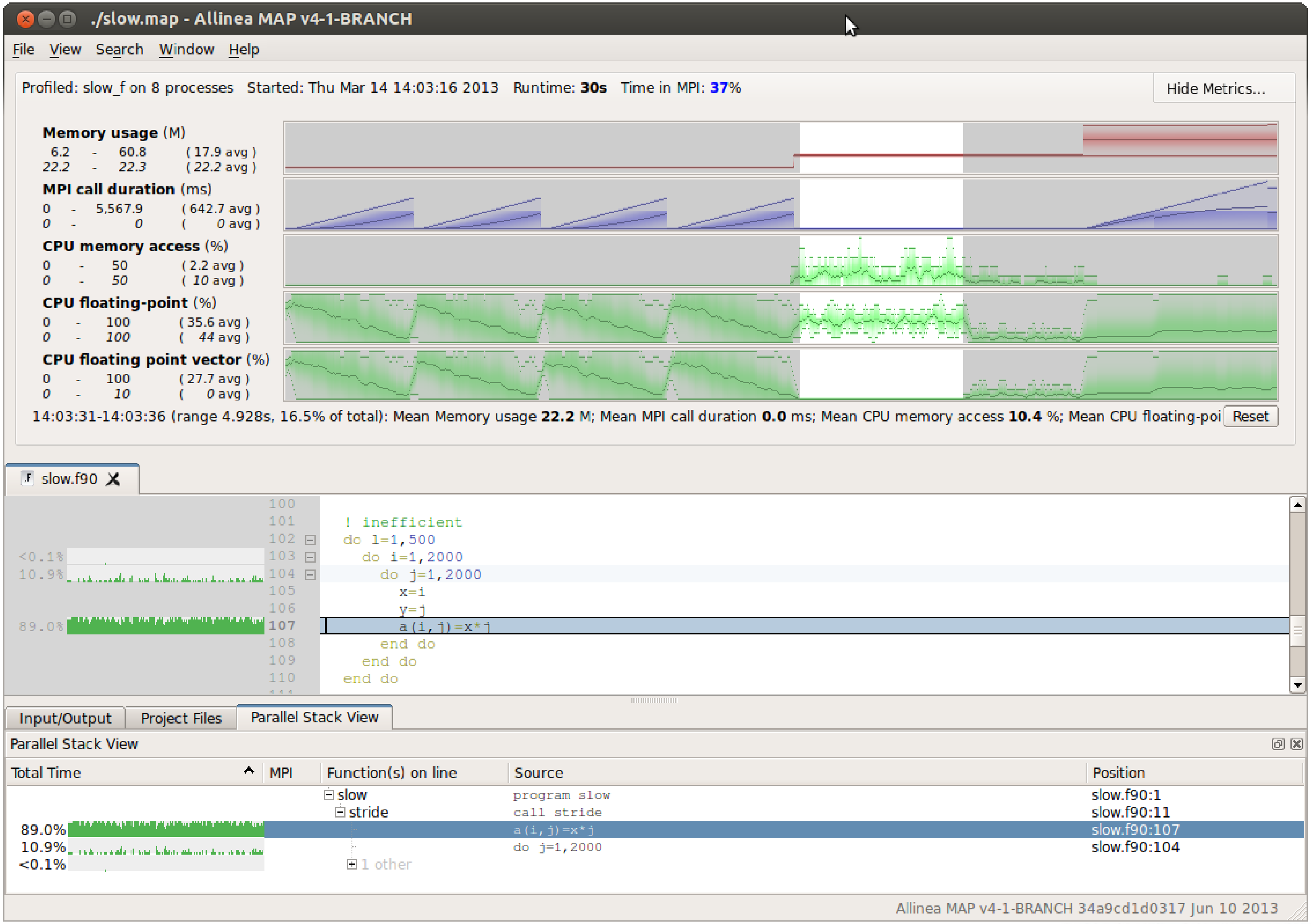
Optimizing for the Xeon Phi
Is my code well-vectorized?

```
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
```

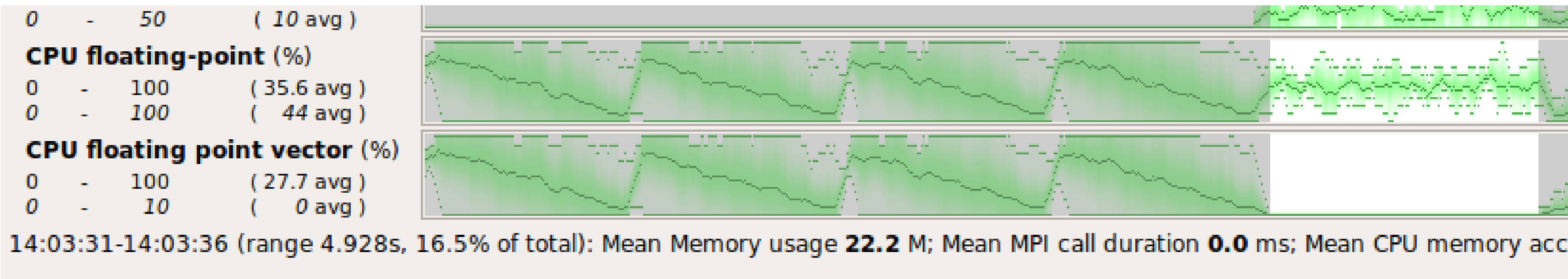
… maybe?

www.allinea.com

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

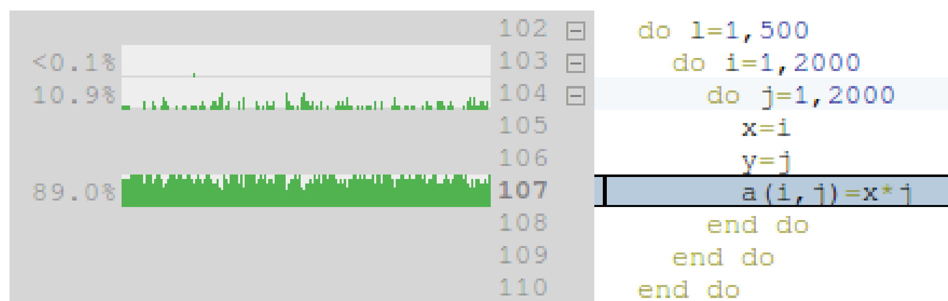


Optimizing for the Xeon Phi
Is my code well-vectorized?

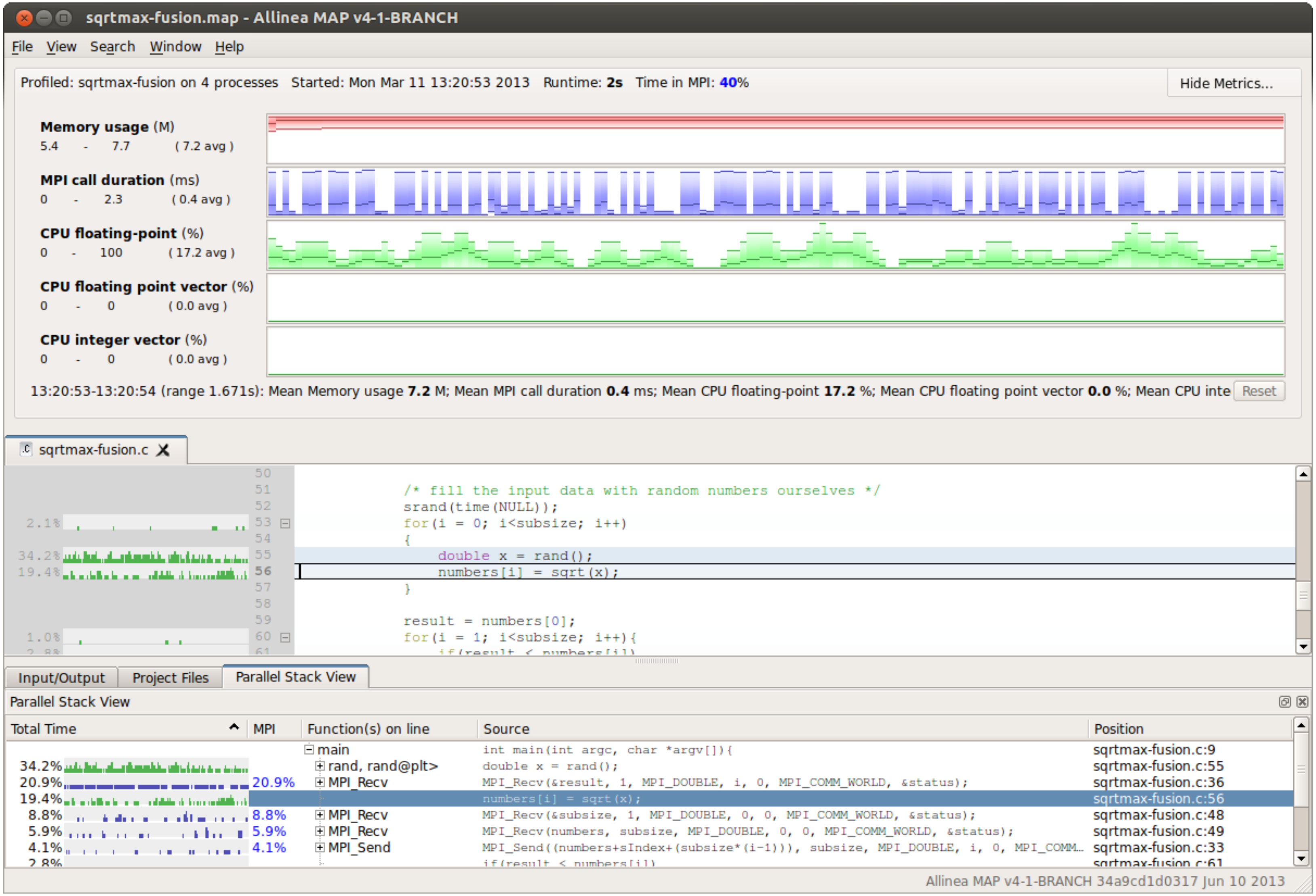


Not in this loop

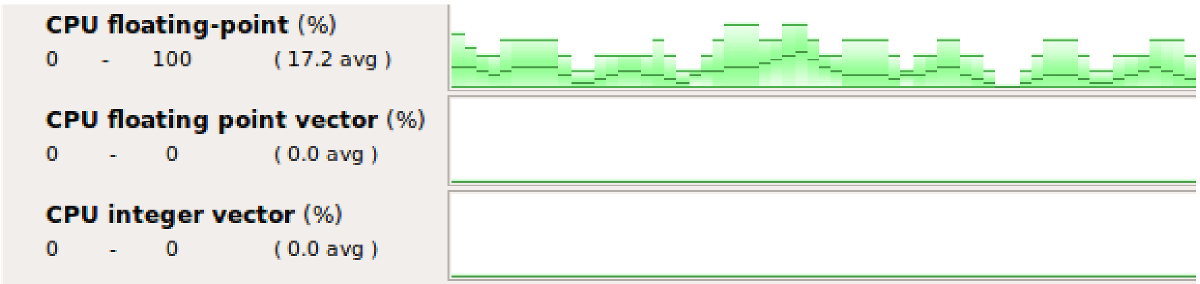
(16.5% of total time)



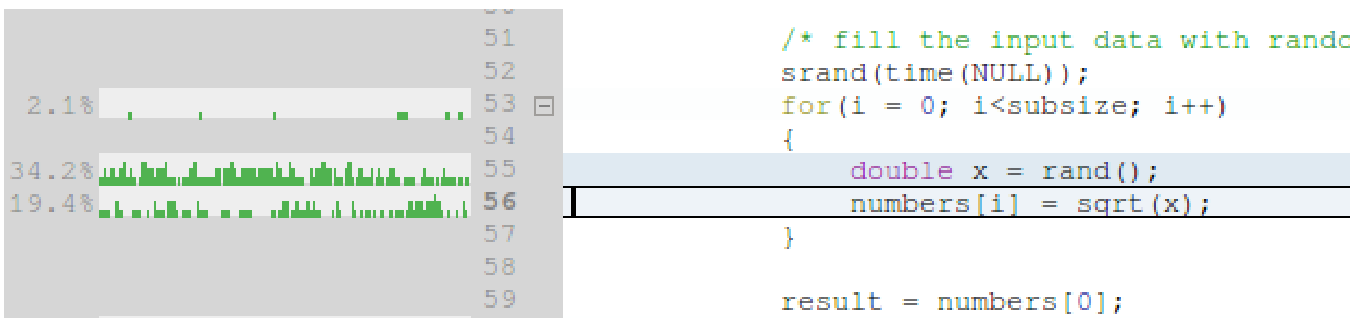
Optimizing for the Xeon Phi
Non-obvious tradeoffs



Optimizing for the Xeon Phi Non-obvious tradeoffs

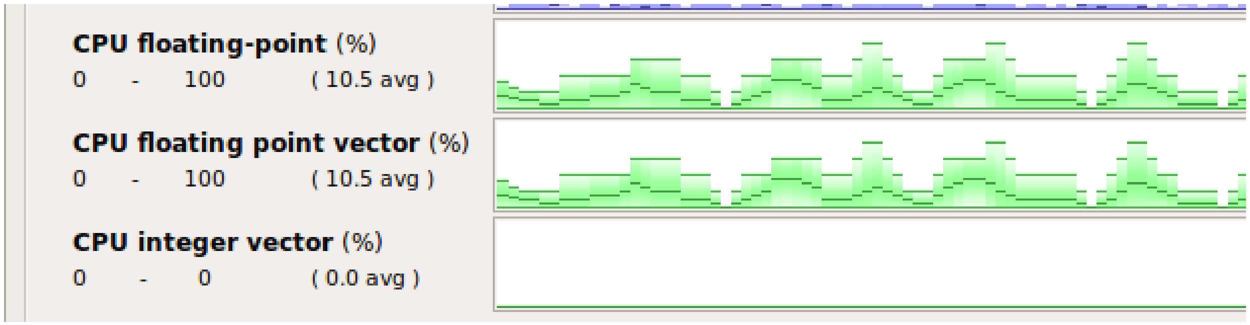


Here a loop taking 55% of total runtime isn't vectorized at all

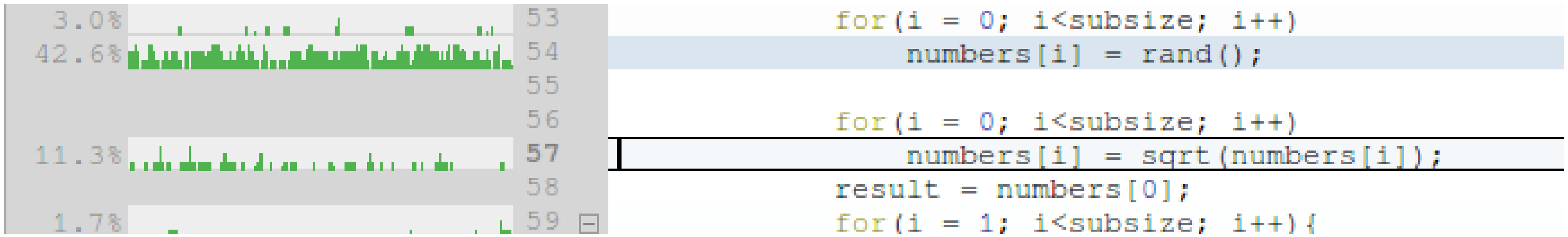


Taking the unvectorizable rand() out of the loop allows the sqrt workload to be fully-vectorized – reverse loop fusion!

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

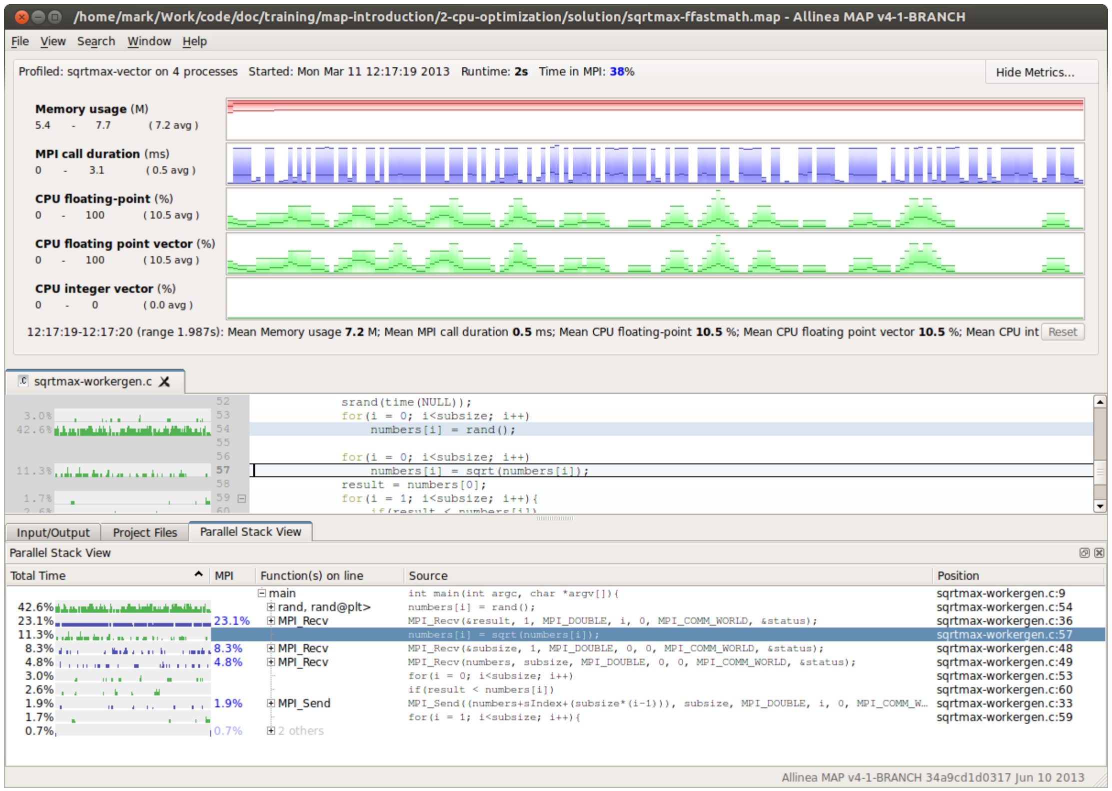
The Intel 13.0 Product Compiler now supports random number auto- vectorization of the drand48 family of random number functions in C/C++ and RANF and Random_Number functions in Fortran. Vectorization is supported through the Intel Short Vector Math Library (SVML).

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

Scalable science needs development tools

HPC is beyond the tipping point for developers

- Print-style debugging cannot cope
- Performance is complex
- Many existing tools failing
- HPC experts are overloaded

Scalable systems need scalable tools

- Tools enable software to exploit the hardware
- Scale does not have to be hard
- Scale does not have to be slow

Allinea is providing the solution

- Allinea DDT and Allinea MAP
- Proven Super-Petascale capable tools
- We understand what HPC developers need

Why tools matter to all of us in HPC…

- "There is an average Ninja gap of 24x", Intel
- "I found a performance problem in just 60 seconds that I've been chasing for 3 weeks"
- "I will show this to my Prof – so we don't waste any more time with Printf"

Three Challenges for tools

Scalability

- Speed and Simplification

Heterogeneity

- Accelerators and Coprocessors

Adoption

- Ease of Use and Education