arm

Meet the experts

Optimize HPC - Application Efficiency on Many-Core Systems

- Florent Lebeau
- 27 March 2018

9 Step guide: optimizing high performance applications

arm

Improving the efficiency of your parallel software holds the key to solving more complex research problems faster. This pragmatic, 9 Step best practice guide will help you identify and focus on application readiness, bottlenecks and optimizations one step at a time.

1 Bugs

- ✔ Correct application.

2 Analyze before you optimize

- ✔ Measure all performance aspects. You can't fix what you can't see.
- ✔ Prefer real workloads over artificial tests.

3 I/O

- ✔ Discover lines of code spending a long time in I/O.
- ✔ Trace and debug slow access patterns.

4 Workload

- ✔ Detect issues with balance.
- ✔ Slow communication calls and processes. Dive into partitioning code.

5 Communication

- ✔ Track communication performance.
- ✔ Discover which communication calls are slow and why.

6 Memory

- ✔ Reveal lines of code bottlenecked by memory access times.
- ✔ Trace allocation and use of hot data structures.

7 Cores

- ✔ Discover synchronization overhead and core utilization.
- ✔ Synchronization-heavy code and implicit barriers are revealed.

8 Vectorization

- ✔ Understand numerical intensity and vectorization level.
- ✔ Hot loops, unvectorized code and GPU performance revealed.

9 Verification

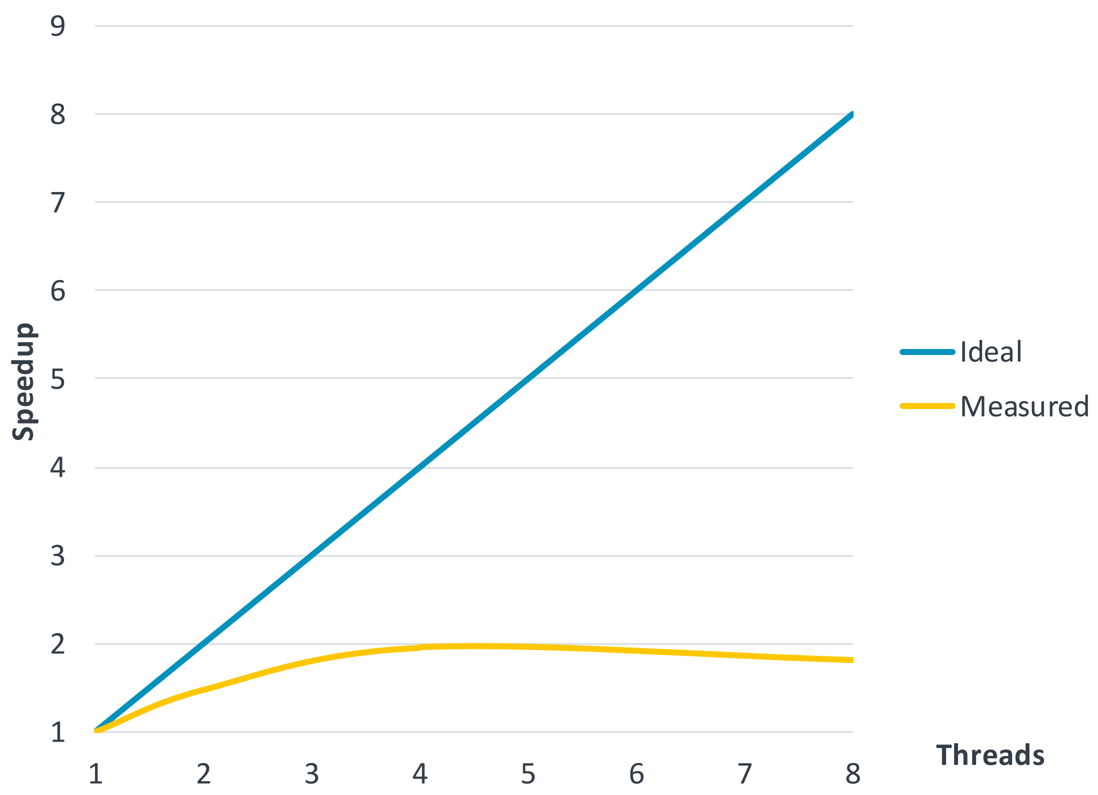
- ✔✔ Validate corrections and optimal Performance.

Key:

- ✔ arm PERFORMANCE REPORTS
- ✔ arm FORGE

Multithreading and scalability

"I wrote my program to run in parallel with a few OpenMP directives… But the performance is not really what I expected."



Example with modified version of Cloverleaf

- Multi-threaded version with OpenMP
- No MPI, no IO

Number of threads	Runtime (seconds)
1	281
2	192
4	144
8	155

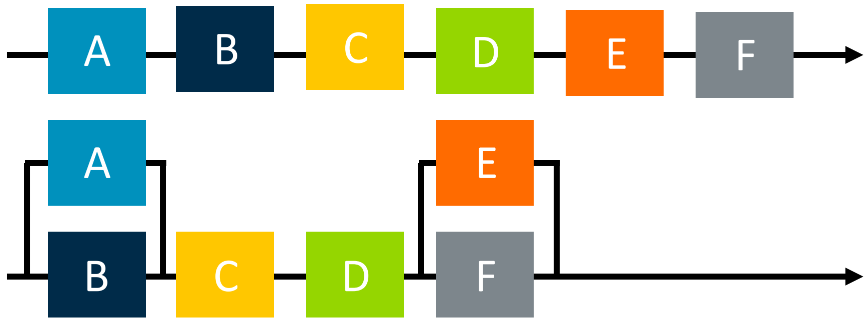
arm

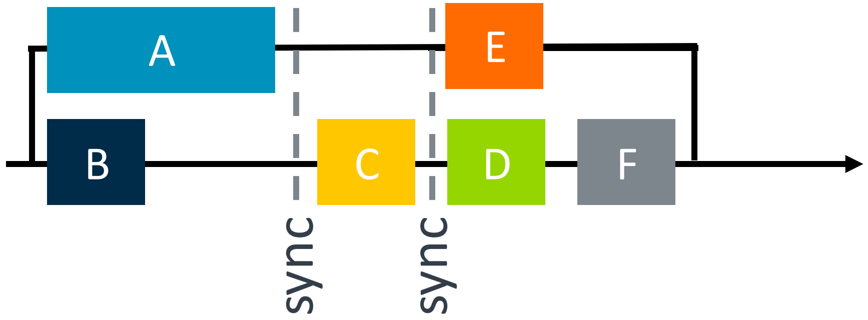
Outline

- What are the challenges of multithreaded applications?
- How to identify issues and act quickly?
- How to understand the performance and optimise the code?

Challenges

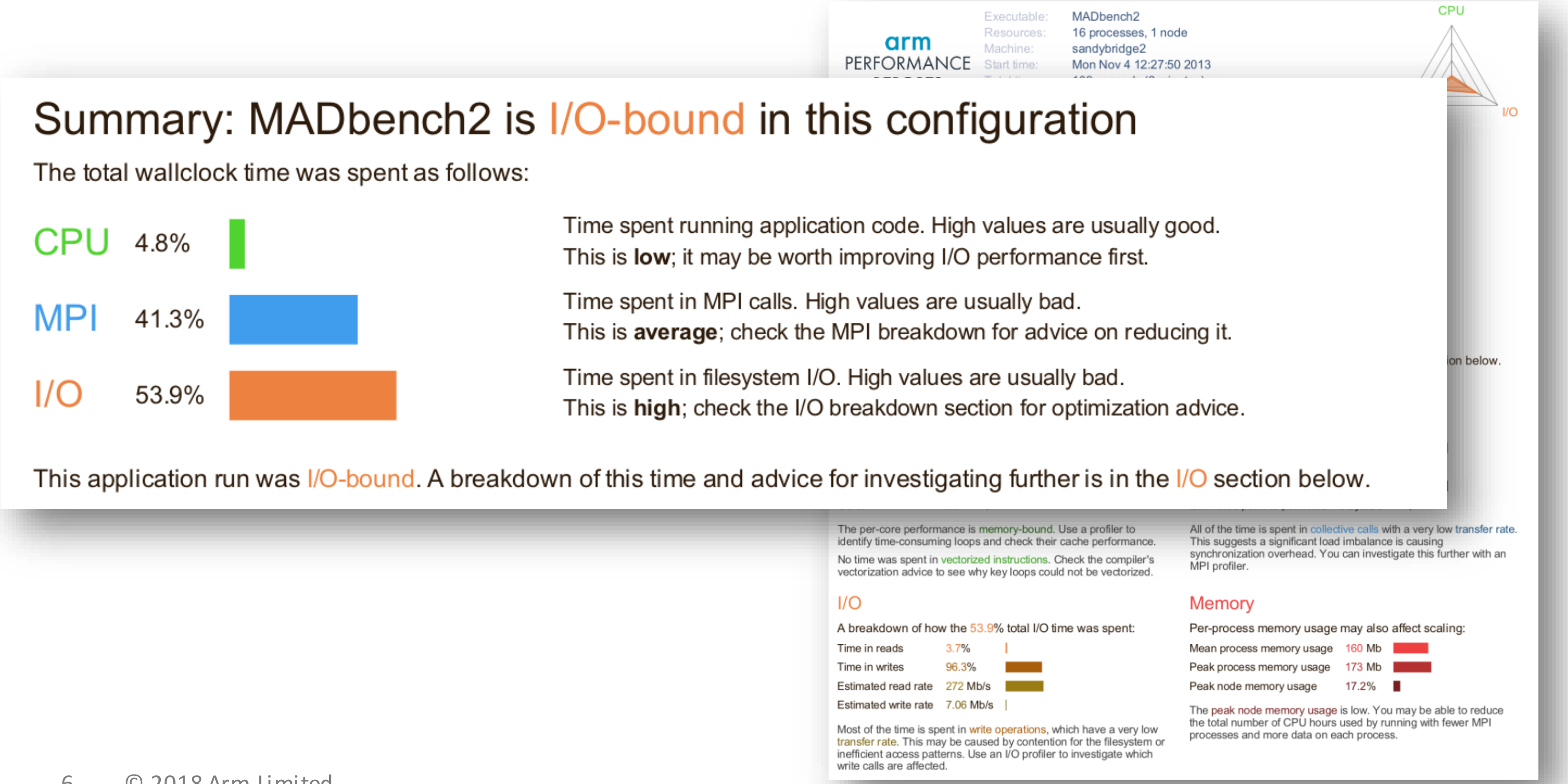
- Sequential sections
  - Regions outside of parallel sections
  - Master or single OpenMP sections
- Synchronisation overhead
  - Load imbalance
  - Implicit and explicit barriers
  - Scheduling policy
  - Communications
  - Hardware contention

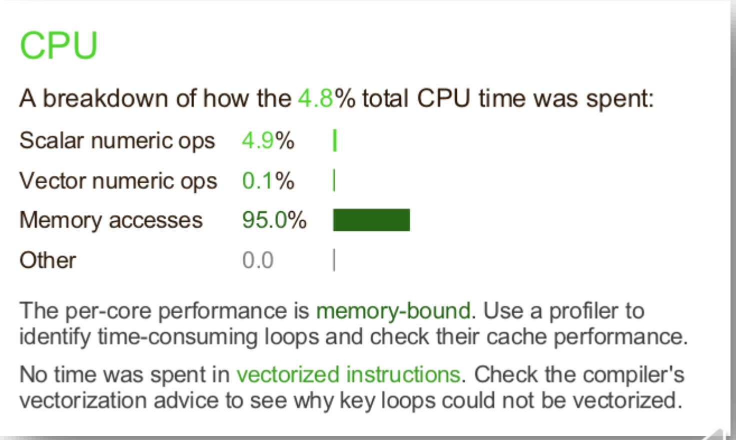




Identifying the amount of sequential code

- Arm Performance Reports is an application reporting tool for HPC
  - Easy to use: no re-compiling required
  - Gives a comprehensible and readable summary of the application behavior

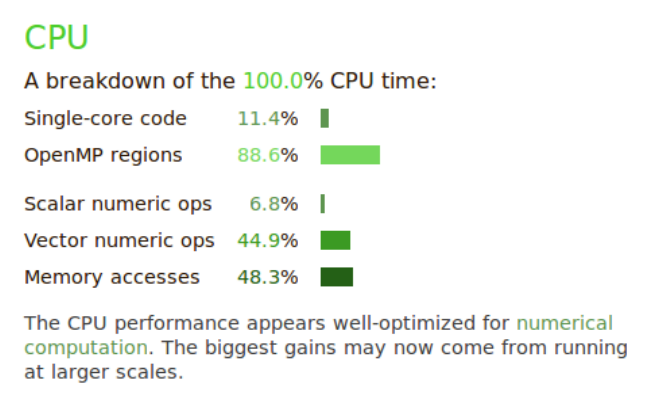


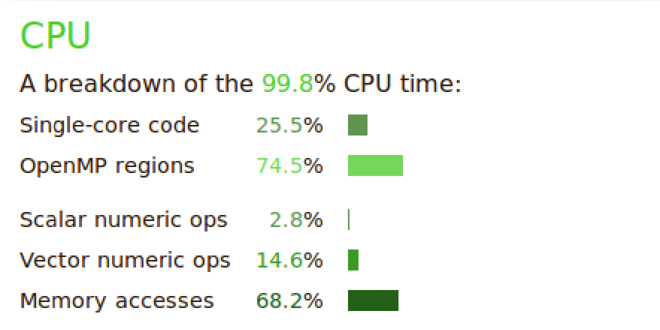


Sequential sections and scalability

- Running Performance Reports on the example using 1 thread indicates that:
  - Time spent in serial code is 11%
  - Theoretical speedup of 4.5 with 8 threads

- With 8 threads:
  - Speedup is only 1.8

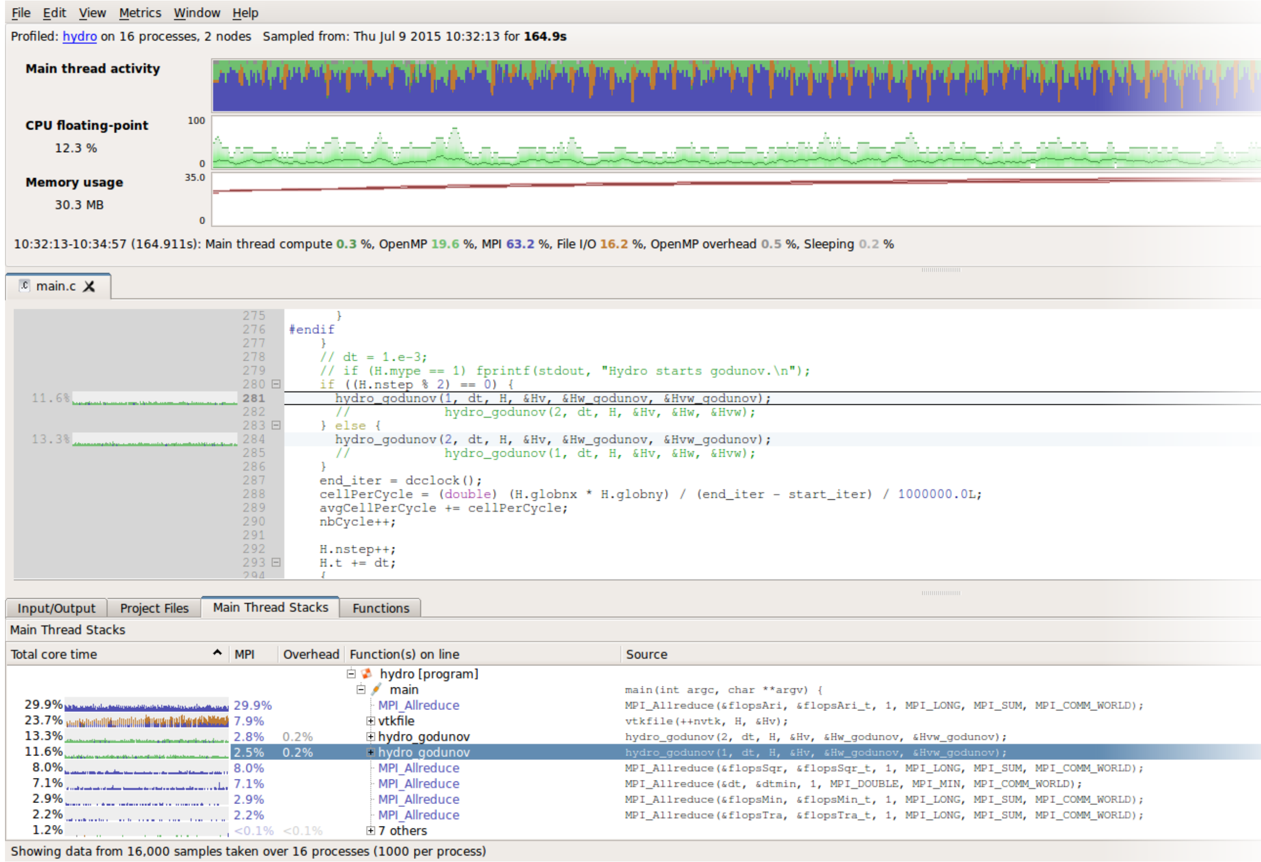




Where does code run serially?

- Arm MAP is a lightweight multi-node profiling tool
  - Compiling with debugging flag required
  - Shows processes and threads activity over time
  - See source code is annotated
  - Information aggregated by stacks and function

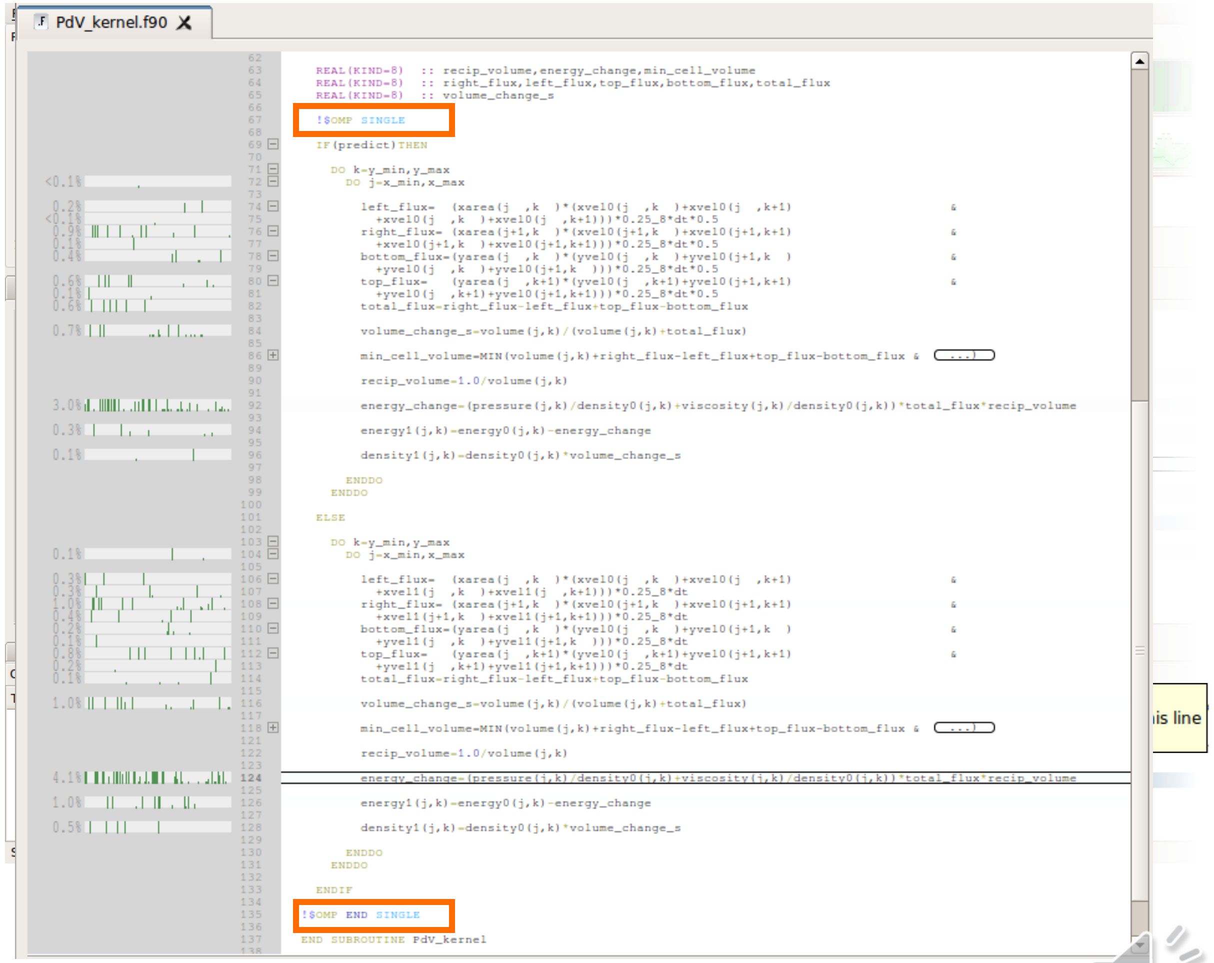
- Compute, IO and MPI



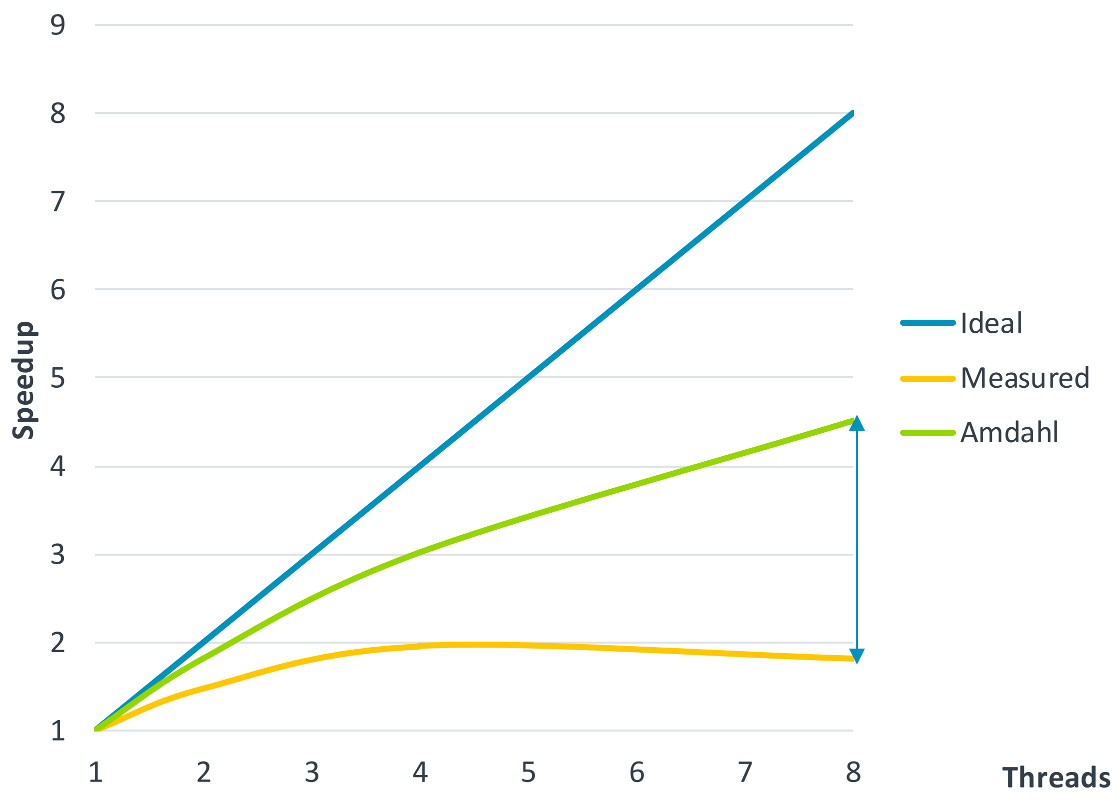
Critical sections

- Profiling our example using 8 threads with Arm MAP shows
  - OpenMP activity in **Light Green**
  - Single thread activity in **Dark Green**
  - Synchronisation in **Grey**

- The tool shows serial execution happens in the *pdv_module::pdv* subroutine
  - In a OpenMP **single** section
  - Can be replaced with OpenMP **parallel do** section



Synchronization overhead



Theoretical speedup: 4.5
Measured: 1.8

Identifying the amount of synchronization

- Performance Reports on the example using 8 threads shows:
  - Low amount of computation

- System load is only 78%

- Possible reasons:
  - Load imbalance
  - Implicit and explicit barriers
  - Scheduling policy
  - Communications
  - Hardware contention

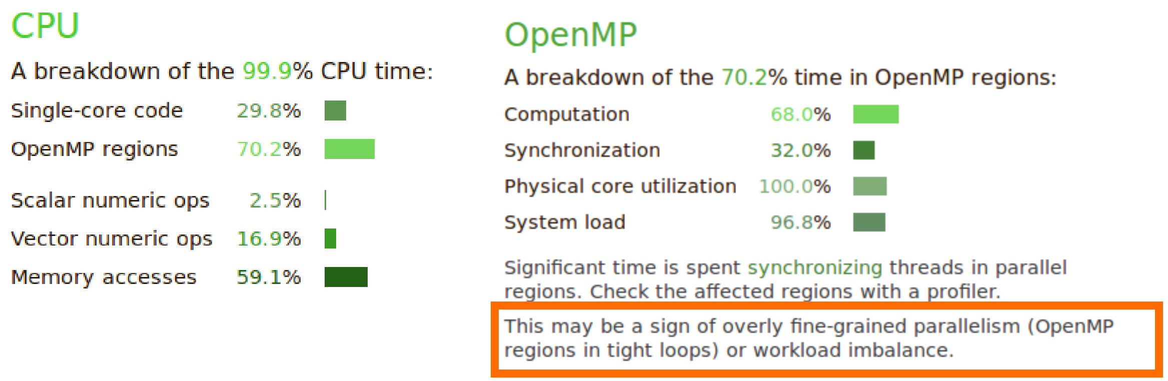
OpenMP

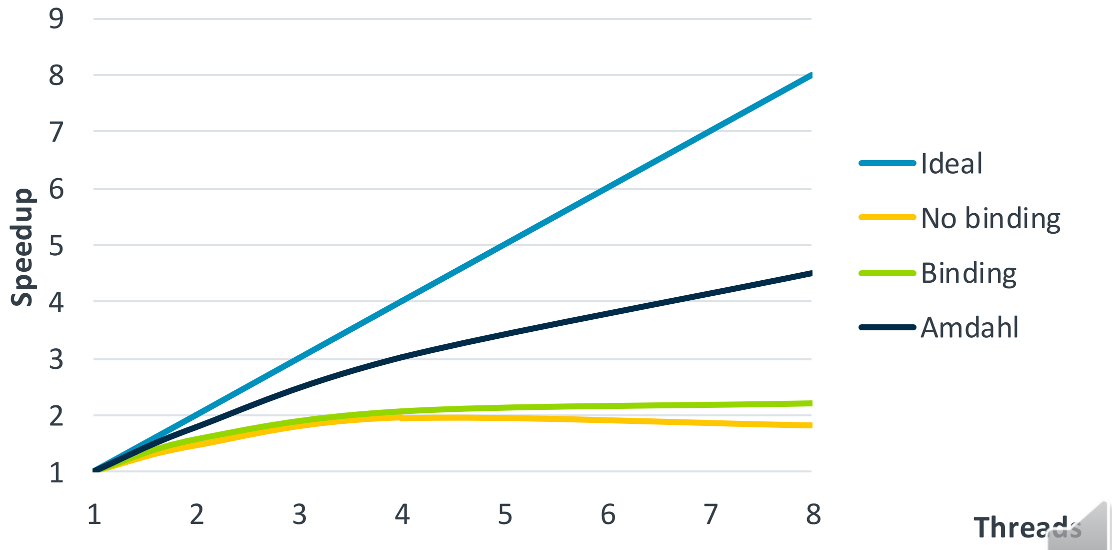
A breakdown of the 74.5% time in OpenMP regions:

Computation	53.6%
Synchronization	46.4%
Physical core utilization	100.0%
System load	78.0%

System Load and thread binding

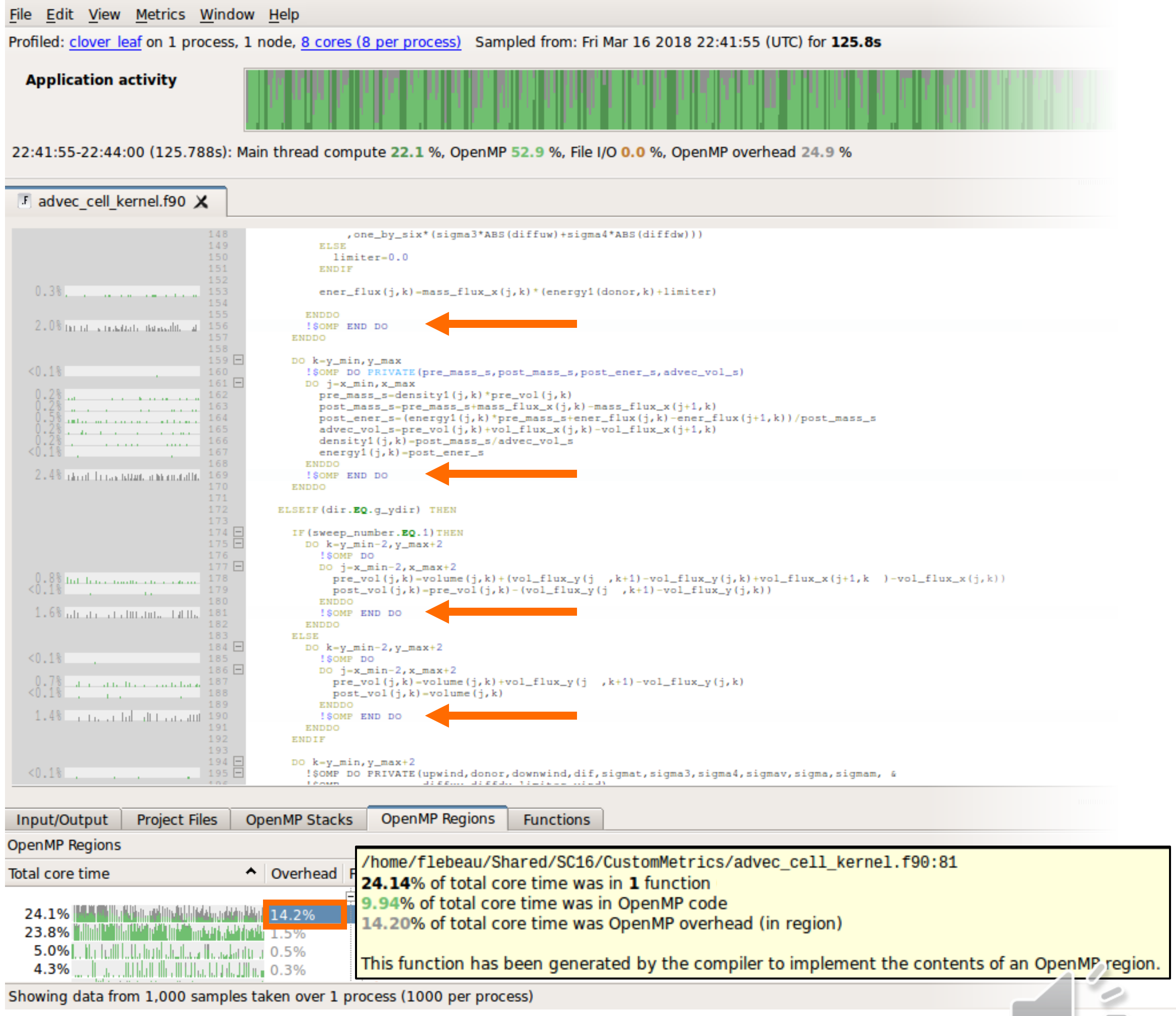
- Dynamic adjustment of the number of threads is enabled
  - OMP_DYNAMIC='TRUE’
- Threads are not bound
  - OMP_PROC_BIND=‘FALSE’
- Binding the threads and disabling dynamic adjustment slightly improve performance
- Performance Reports detects a sign of overly fine-grained parallelism





Where are threads waiting?

- Profiling Cloverleaf using 8 threads (bound to cores) with Arm MAP shows
  - Overhead in one OpenMP region
- Implicit barriers
  - Inner loop parallelization only

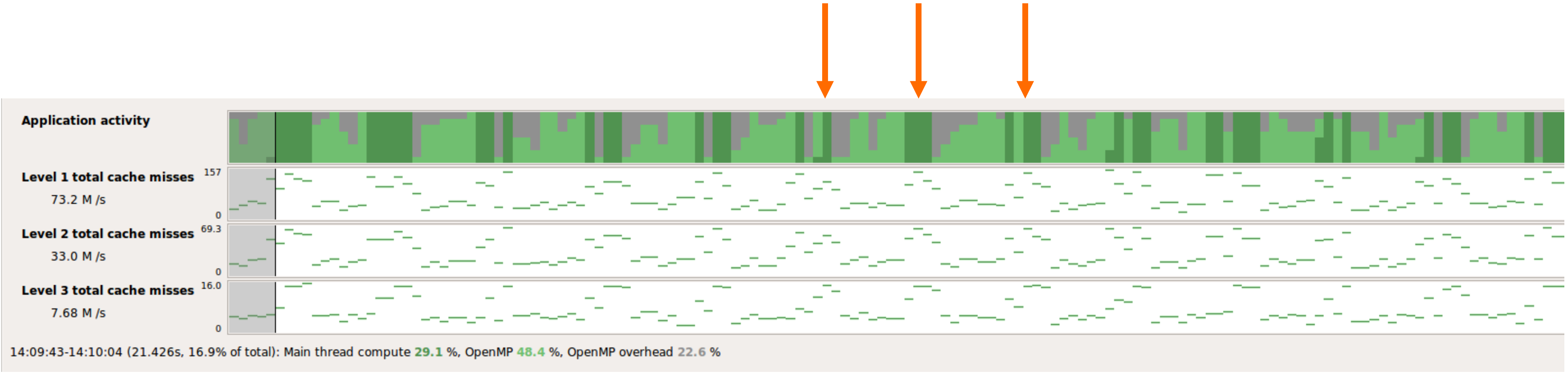


Understanding resource usage

- Memory accesses

Additional metrics

- PAPI



Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
תודה

arm

