Language-Centric Performance Analysis of OpenMP Programs with Aftermath

Andi Drebes

The University of Manchester School of Computer Science Advanced Processor Technologies andi.drebes@manchester.ac.uk

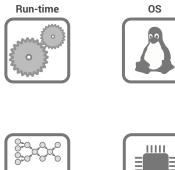
Joint work with:

Jean-Baptiste Bréjon, Antoniu Pop, Karine Heydemann, Albert Cohen

IWOMP 2016



Analysis of OpenMP Programs



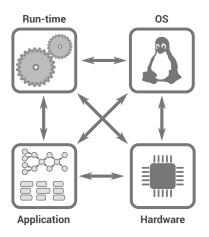


Application

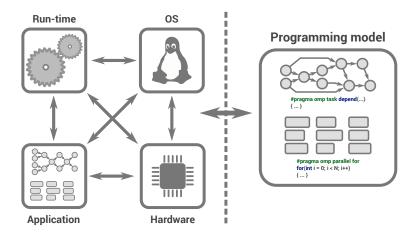


Hardware

Analysis of OpenMP Programs



Analysis of OpenMP Programs



New Tools for Performance Analysis

Frequent topics for performance analysis:

- Amount of parallelism and load balacing
- Duration of execution phases
- Synchronization overhead (e.g., barriers)
- Choice of an appropriate loop schedule
- Data distribution on NUMA systems
- Relate hardware events to loops / tasks

New Tools for Performance Analysis

Frequent topics for performance analysis:

- Amount of parallelism and load balacing
- Duration of execution phases
- Synchronization overhead (e.g., barriers)
- Choice of an appropriate loop schedule
- Data distribution on NUMA systems
- Relate hardware events to loops / tasks

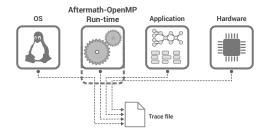
Our tools: Aftermath & Aftermath-OpenMP

- Aftermath: Graphical tool for performance analysis
- Aftermath-OpenMP. Instrumented LLVM/clang run-time

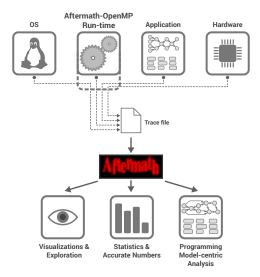
Outline

- 1. Overview of Trace-based Analysis
- 2. Overview of Aftermath's GUI
- 3. Demo
- 4. Overhead of Tracing
- 5. Summary & Conclusion

Trace-based Analysis with Aftermath



Trace-based Analysis with Aftermath



```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{ ... }</pre>
```

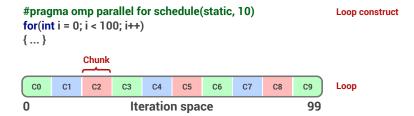
```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{ ... }</pre>
```

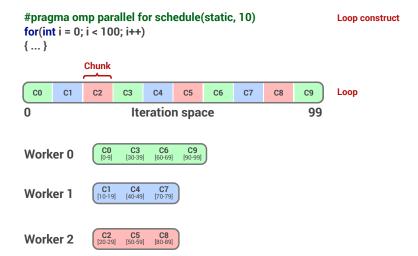
Loop construct

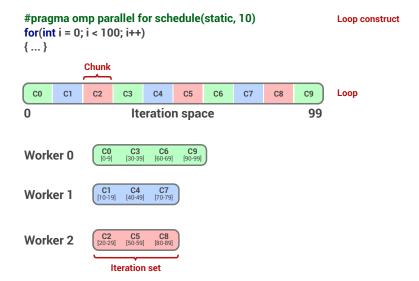
```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
{...}
Loop
0 Iteration space 99</pre>
```

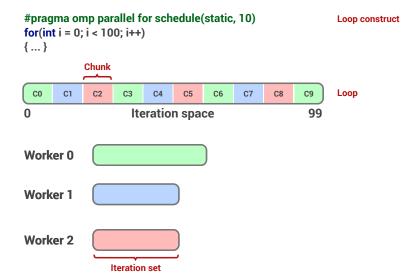
```
#pragma omp parallel for schedule(static, 10) Loop construct
for(int i = 0; i < 100; i++)
{ ... }</pre>
```

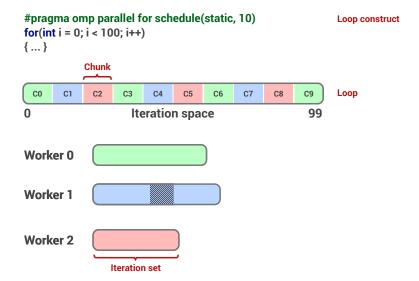


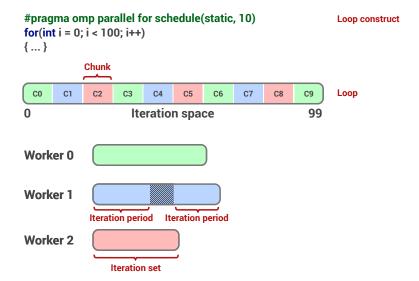


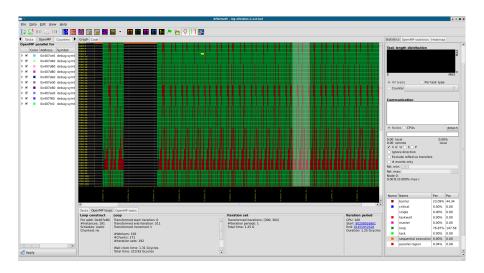


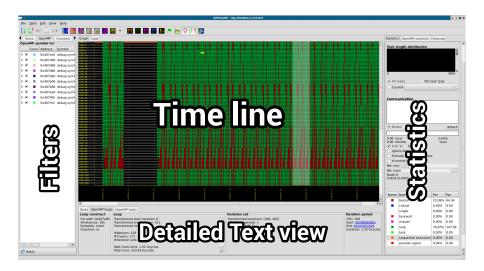


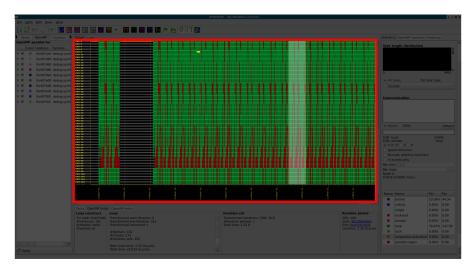










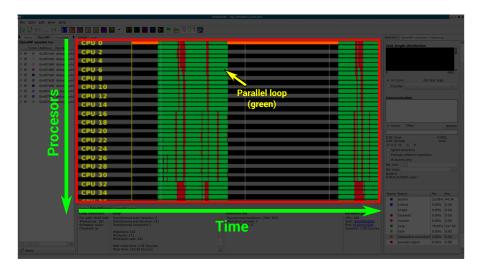


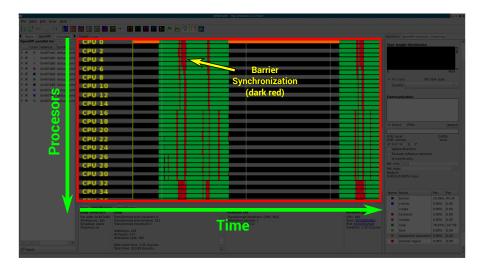
Time line

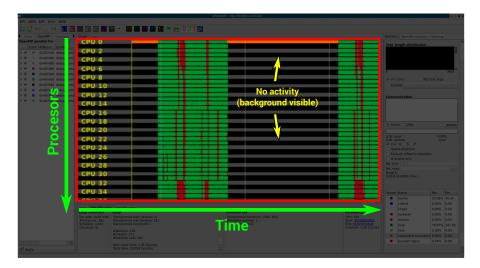


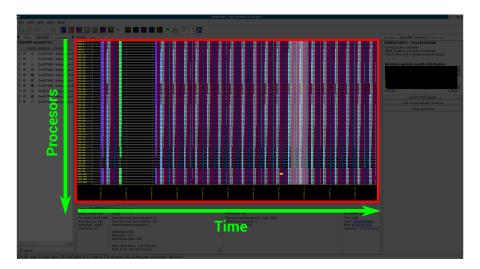
Time line







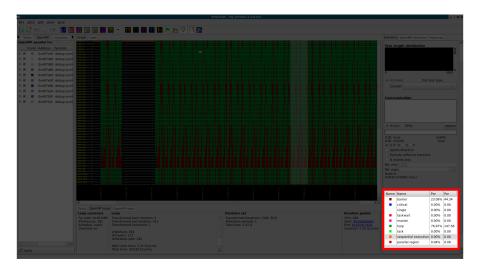




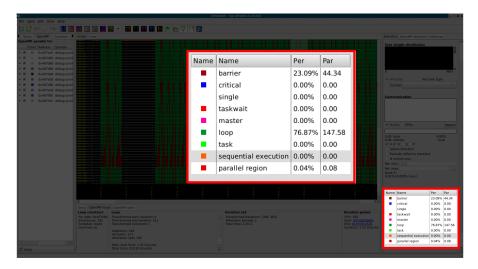
Time line: Loop constructs

14		Aftermath + mg-alistates.C.ost.bz2		
Ble Data Edit View Help				
		/* 🗠 🛛 🗖 🔛		
Tasks OpenMP Counters OpenMP parallel for	Graph Code		Statistics OpenMP statistics Heat 94462174975 - 101198193468	
Color Address Symb	CPU 2			
🕨 🔳 0x407ce0 debu; yml	CPU 4		9589 Iteration periods considered 101.87 M cycles / iteration period	(avg)
▶ Ø = 0x407d40 debug yml ▶ Ø = 0x407d50 debug yml			Iteration periods length distrib	witten
N ■ 0x407d80 debu; yml	CPU 6			
🕨 🗷 🔳 0x407da0 debug yml	CPU 8			
▶ ₩ ■ 0x407e00 debut yml ▶ ₩ ■ 0x407e80 debut yml	CPU 10			
🕨 🔳 0x407e 🖉 stor yml	CPU 12		273.00	1.30 G
► K ■ 0x407fNg debut ymi ► K ■ 0x407fgg debut ymi	CPU 14		Select from graph	
Proceso	CPU 16			
	CPU 18			
	CPU 20			
	CPU 22			
	CPU 24			
	CF0 20			
	CPU 28			
	CPU 30			
	CPU 32			
	CPU 34			
	CPU 36			
	CPU 38			
	Tarke OnentiPlanes OnentiPlants			
	For addr: 0x407e80 Transformed start iteration: 0	Transformed iterations: [300, 302]	CPU: 100	
		Time		
	Chunked: no #Workers: 192	inne		
	#Chunks: 171 #Iteration sets: 192			
	Wall clock time: 1.31 Gcycles			

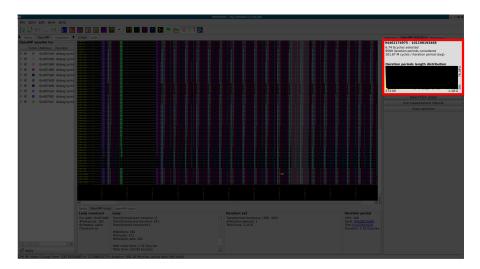
Time line: Loop constructs



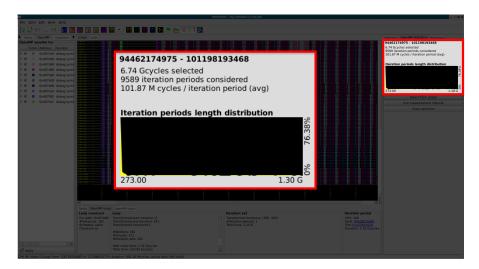
State statistics



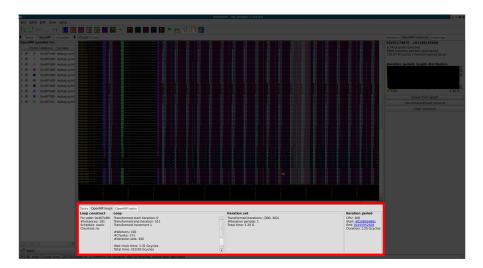
State statistics



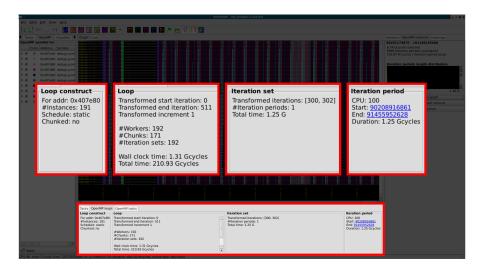
Histogram showing duration of iteration periods



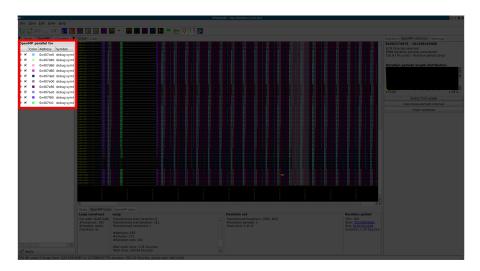
Histogram showing duration of iteration periods



Detailed text view for parallel loops

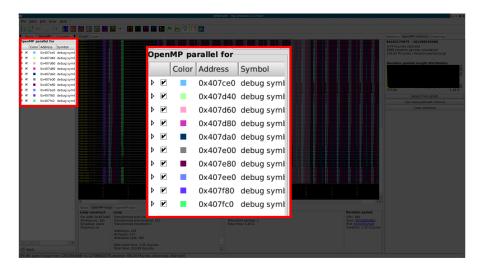


Detailed text view for parallel loops



Filter for loop constructs

Aftermath: Overview of the GUI



Filter for loop constructs

Benchmark: NPB MG

- ▶ NPB 2.3 C implementation from the Omni Compiler Project
- ► C input class (512 × 512 elements)

Test platform

- SGI UV 2000 (Xeon E5-4640)
- 192 cores (Hyperthreading disabled)
- > 24 NUMA nodes, 756 GiB RAM
- LLVM/clang 3.8.0
- Aftermath-OpenMP for trace generation

DEMO

Execution phases

- Parallel initializations + Main Computation
- Sequential execution in between

Execution phases

- Parallel initializations + Main Computation
- Sequential execution in between

Time spent in barriers

States on time line / statistics panel

Execution phases

- Parallel initializations + Main Computation
- Sequential execution in between

Time spent in barriers

States on time line / statistics panel

Load imbalance

- Sufficient parallelism
- High load imbalance, but not due to partitioning / schedule
- ▶ Same NUMA node \rightarrow Aprox. same execution time

Execution phases

- Parallel initializations + Main Computation
- Sequential execution in between

Time spent in barriers

States on time line / statistics panel

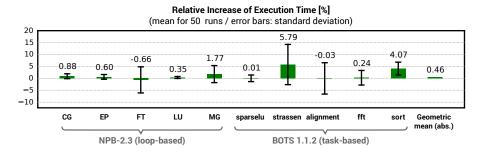
Load imbalance

- Sufficient parallelism
- High load imbalance, but not due to partitioning / schedule
- ▶ Same NUMA node \rightarrow Aprox. same execution time

Solution

- Change allocation scheme: one big allocation
- ▶ Reduce number of workers: #iters = *n* × #workers
- ► Result: 35× speedup

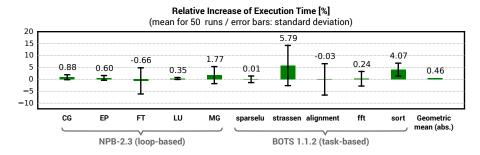
Overhead of Tracing



Test system

SGI UV 2000 (192 cores, 24 NUMA nodes)

Overhead of Tracing



Test system

SGI UV 2000 (192 cores, 24 NUMA nodes)

Missing benchmarks

- Outlier. floorplan (+380% execution time; very small tasks)
- Segfaults (*BT*, *nqueens*, *uts*) / Excessive Execution time (*IS*) / Verification Failure (*health*)

Using Aftermath & Aftermath-OpenMP

Drop-in replacement for libomp with wrapper script:

- \$ aftermath-openmp-trace -o events.ost -- <program> <args>
- \$ aftermath events.ost

Using Aftermath & Aftermath-OpenMP

Drop-in replacement for libomp with wrapper script:

- \$ aftermath-openmp-trace -o events.ost -- <program> <args>
- \$ aftermath events.ost

Source code and tutorial:

http://www.openstream.info/aftermath

Virtual Machine (Aftermath + Aftermath-OpenMP + sample traces + documentation): http://www.openstream.info/vm

Aftermath

- ► Reactive graphical user interface for trace analysis
- Programming model-centric analysis: Loops and tasks

Aftermath

- ► Reactive graphical user interface for trace analysis
- Programming model-centric analysis: Loops and tasks

Aftermath-OpenMP

- Instrumented LLVM/clang OpenMP run-time
- Low tracing overhead

Aftermath

- ► Reactive graphical user interface for trace analysis
- Programming model-centric analysis: Loops and tasks

Aftermath-OpenMP

- Instrumented LLVM/clang OpenMP run-time
- Low tracing overhead

Future work

- Dependent tasks
- Automate recurring analyses

Aftermath

- ► Reactive graphical user interface for trace analysis
- Programming model-centric analysis: Loops and tasks

Aftermath-OpenMP

- Instrumented LLVM/clang OpenMP run-time
- Low tracing overhead

Future work

- Dependent tasks
- Automate recurring analyses

On-line resources

http://www.openstream.info/aftermath (Main website) http://www.openstream.info/vm (VM image)