

OpenMP Programming Workshop @LRZ

The  *Common Core and Beyond*
Enabling HPC since 1997

Manuel Arenaz | February 11-13, 2020

©Appentra Solutions S.L.



Agenda

8:30 - 9:00

Setup and welcome participants

9:00 - 9:15

Overview

9:15 - 10:30

The OpenMP Common Core

Decomposing code into patterns for parallelization

Using Parallelware Trainer: A walk-through with PI example

10:30 - 11:00

Coffee

11:00 - 12:40

Practicals: Examples codes PI, MANDELBROT, HEAT and LULESHmk

Worksheet: Parallelizing PI and LULESHmk with OpenMP

12:40 - 13:00

Close

Hands-on Lab

- Install and launch Parallelware Trainer:

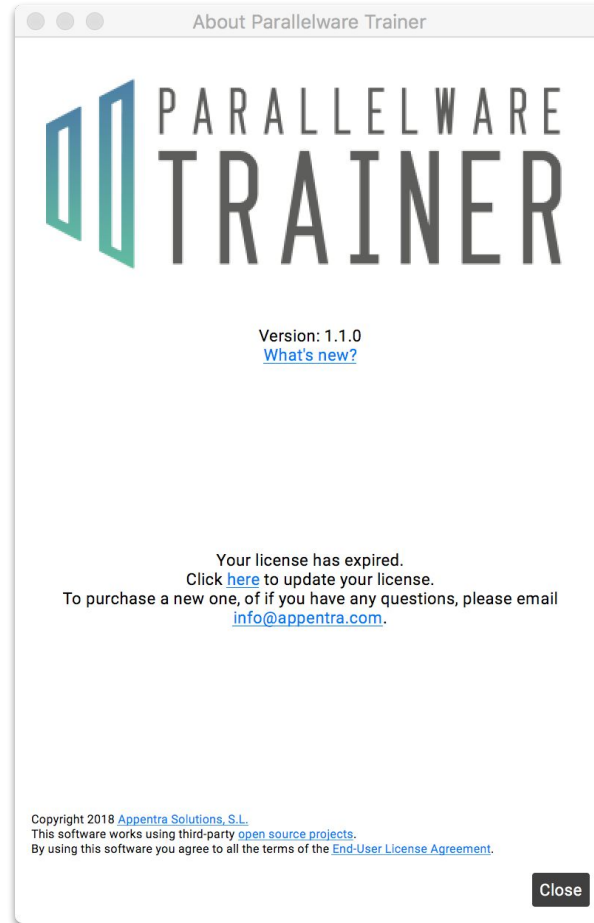
<https://www.appentra.com/>

- Practicals:
 - Parallelizing the calculation of π
 - Parallelizing a micro-kernel of the CORAL-lulesh benchmark
 - Parallelizing HEAT and Mandelbrot.

★ Follow the instructions on the worksheet



A screenshot of the Appentra website homepage. The top navigation bar includes the Appentra logo (a red and white circle) and the tagline "make code parallel", followed by menu items: PRODUCTS, TECHNOLOGY, TRAINING, APPENTRA, NEWS, and FEDER. The main content area features the heading "Parallelware Trainer" in large black font. Below it is the subtext "THE HPC LEARNING TOOL TO HELP YOU LEARN ABOUT HPC AND GET YOUR JOB DONE!". There are two buttons: a blue "Try Parallelware Trainer" button and a white "Or contact us" button. On the right side, there is a large image showing a code editor interface with a file explorer on the left and code in the main area, set against a light blue and green circular background. The Parallelware Trainer logo is also visible in the bottom right corner of the image area.



Parallelware Trainer

Project Explorer

Code Editor

Version Manager

The screenshot displays the Parallelware Trainer IDE interface. At the top, there are three tabs: "Project Explorer", "Code Editor", and "Version Manager". The "Project Explorer" on the left shows a project named "pi" with files "Makefile", "pi.c", and "README.md". The "Code Editor" in the center shows the source code for "pi.c", which includes a main function and a "getClock" function. The code uses OpenMP for parallelization. The "Version Manager" on the right shows a comparison between the original code and a modified version. The bottom panel is the "Output Consoles", which shows the results of a parallelization analysis. The analysis identifies several parallelization opportunities, such as scalar reduction and loop parallelization, and reports that the parallelization completed successfully.

```
pi.c
35
36 out_result = 4.0 / N * sum;
37
38 // =====
39 double time_finish = getClock();
40
41 // Prints an execution report
42 printf("time (s)= %.6f\n", time_finish - time_start);
43 printf("result\t= %.8f\n", out_result);
44 const double realPiValue = 3.141592653589793238;
45 printf("error\t= %.1e\n", fabs(out_result - realPiValue));
46
47 return 0;
48 }
49
50 double getClock() {
51 #ifdef _OPENMP
52 #include <omp.h>
53 return omp_get_wtime();
54 #elif __linux__ || __APPLE__
55 #include <time.h>
56 struct timespec ts;
57 clock_gettime(CLOCK_MONOTONIC, &ts);
58 return ts.tv_sec + ts.tv_nsec / 1.0e9;
59 #else
60 #include <time.h>
61 return (double)clock() / CLOCKS_PER_SEC;
62 #endif
63 }
64
```

```
Original 1
32 out_result = 4.0 / N * sum;
33
34 // =====
35 double time_finish = getClock();
36
37 // Prints an execution report
38 printf("time (s)= %.6f\n", time_finish - time_start);
39 printf("result\t= %.8f\n", out_result);
40 const double realPiValue = 3.141592653589793238;
41 printf("error\t= %.1e\n", fabs(out_result - realPiValue));
42
43 return 0;
44 }
45
46 double getClock() {
47 #ifdef _OPENMP
48 #include <omp.h>
49 return omp_get_wtime();
50 #elif __linux__ || __APPLE__
51 #include <time.h>
52 struct timespec ts;
53 clock_gettime(CLOCK_MONOTONIC, &ts);
54 return ts.tv_sec + ts.tv_nsec / 1.0e9;
55 #else
56 #include <time.h>
57 return (double)clock() / CLOCKS_PER_SEC;
58 #endif
59 }
60
```

```
[12:05:56] Parallelizing...
pi.c line 27: Parallel scalar_reduction pattern identified for variable 'sum' with associative, commutative operator '+'
pi.c line 27: Available parallelization strategies for variable 'sum'
pi.c line 27: #1 OpenMP scalar_reduction (+ implemented)
pi.c line 27: #2 OpenMP atomic access
pi.c line 27: #3 OpenMP explicit privatization
pi.c line 27: Loop parallelized with multithreading using OpenMP directive 'for'
pi.c line 27: Parallel_region defined by OpenMP directive 'parallel'
pi.c line 27: Make sure there is no aliasing among arguments in 'main': argc, argv
[12:05:56] Parallelization completed successfully
[12:05:57] Analysis completed: 0 opportunities found
```

Output Consoles

Parallelizing PI & LULESHmk



Goals:

- Follow the instructions in the worksheet for PI and LULESHmk.
 - Additional case studies: HEAT and MANDELBROT.
- Understand parallelization guided by code patterns.
- Generate, build and run multiple versions using Parallelware Trainer.

PI: Parallel scalar reduction

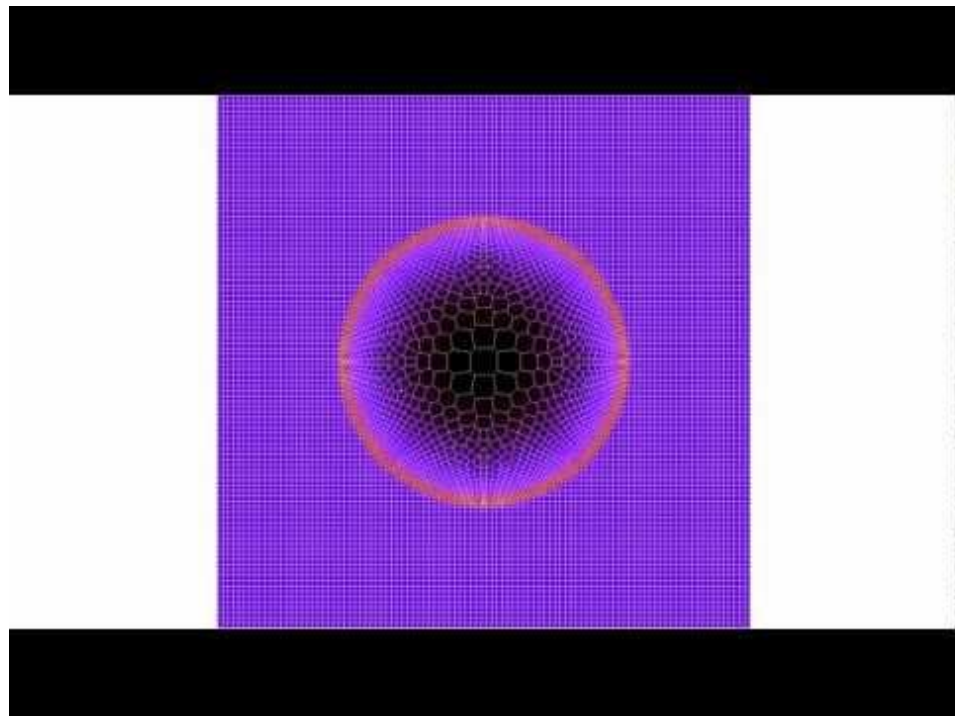
Code file "pi.c"		Pattern			
Function	Line	Forall	Scalar reduction	Sparse reduction	Convergence loop
main()	29		sum		

LULESHmk: Parallel Sparse Reduction

Livermore **U**nstructured **L**agrange
Explicit **S**hock **H**ydrodynamics

Part of a Physics Simulation
software (ALE3D)

Models the propagation of a Sedov blast
wave using Lagrangian hydrodynamics



How to verify correctness of LULESHmk

```
$ ./luleshmk
- Configuring the test...
- Executing the test...
gprof ./luleshmk
- Verifying the test...
Run completed:
  Problem size      = 30
  MPI tasks        = 1
  Iteration count   = 932
  Final Origin Energy = 1.000000e+00
  Number of nodes   = 27000
  Number of elements = 30000
  Number of regions = 1
    Region 1 of size 30000
  Testing Plane 0 of Energy Array on rank 0:
    MaxAbsDiff      = 8.410000e+02
    TotalAbsDiff    = 1.303550e+05
    MaxRelDiff      = 9.655568e-01

Elapsed time      = 71.00 (s)
Grind time (us/z/c) = 2.821491 (per dom) ( 2.821491
overall)
FOM               = 354.42254 (z/s)
```

Profiling of LULESHmk

```
$ gcc -pg -o luleshmk luleshmk.c -lm
$ ./luleshmk
$ gprof ./luleshmk
```

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
56.23	24.62	24.62	223680000	0.00	0.00	CalcElemFBHourglassForce_workload
22.76	34.59	9.97	932	0.01	0.01	ApplyMaterialPropertiesForElems_workload
15.26	41.27	6.68	27960000	0.00	0.00	CalcElemVelocityGradient_workload
2.26	42.26	0.99	932	0.00	0.03	CalcFBHourglassForceForElems
2.24	43.24	0.98	27960000	0.00	0.00	CalcElemFBHourglassForce
0.75	43.57	0.33	27960000	0.00	0.00	CalcElemVelocityGradient
0.34	43.72	0.15	1	0.15	43.76	luleshmk
0.09	43.76	0.04	932	0.00	0.01	CalcKinematicsForElems
0.09	43.80	0.04				frame_dummy
0.00	43.80	0.00	932	0.00	0.01	ApplyMaterialPropertiesForElems
0.00	43.80	0.00	2	0.00	0.00	calculate_checksum
0.00	43.80	0.00	2	0.00	0.00	getClock
0.00	43.80	0.00	1	0.00	0.00	Parameters_create
0.00	43.80	0.00	1	0.00	0.00	Parameters_free
0.00	43.80	0.00	1	0.00	0.00	VerifyAndWriteFinalOutput

Decomposition of LULESHmk into patterns

Code file "luleshmk.c"		Pattern				
Function	Line	Forall	Scalar reduction	Sparse reduction	Convergence loop	
CalcElemFBHourglassForce_workload()	60		sum			
CalcElemFBHourglassForce()	73	hgfx				
CalcFBHourglassForceForElems()	131			domain_m_fx		
ApplyMaterialPropertiesForElems_workload()	187					
ApplyMaterialPropertiesForElems()	210 217					
CalcElemVelocityGradient_workload()	231					
CalcKinematicsForElems()	258					
luleshmk()	292				iter (loop index)	
main()	349					
	358					
	365					
VerifyAndWriteFinalOutput()	485					