Performance Analysis
and
Shared Memory Parallelisation of FDS

Fire and Evacuation Modeling Technical Conference 2014
Gaithersburg, September 8-10 2014

Daniel Haarhoff, Lukas Arnold

email: l.arnold@fz-juelich.de

Jülich Supercomputing Centre
Institute for Advanced Simulation
Forschungszentrum Jülich GmbH, Germany

Provokative Motivation

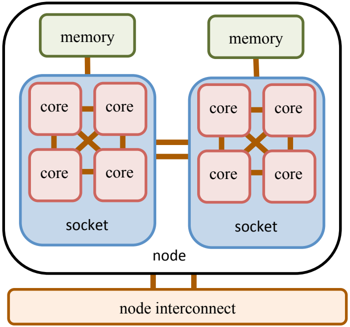
Q: Why did you use THIS grid resolution?

A: To have the simulation done by tomorrow.

Motivation and Computer Architecture

Why shared memory / hybrid parallelisation in FDS?

- make use of more hardware resources
- soften resource limitation due to mesh boundary placement



Motivation for hybrid parallelisation

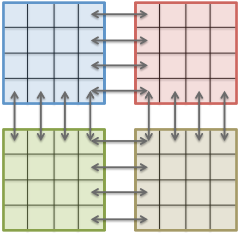
- hierarchical communication structures
- non-uniform memory access

Parallelisation approach

- OpenMP on compute nodes / sockets (shared memory)
- MPI for inter-node communication (distributed memory)

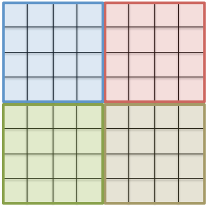
Selected Programming Modells

MPI



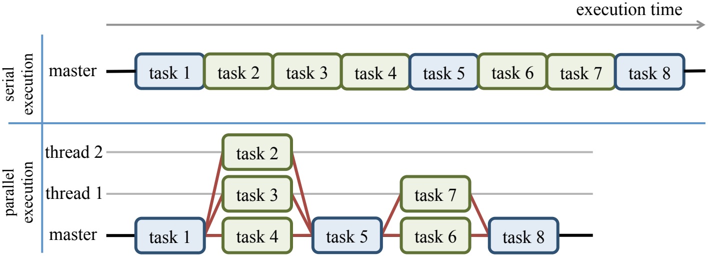
- dispach of processes, each assigned a rank
- explicit communication of boundaries / memory managed by programmer
- needs change of algorithms and data structures

OpenMP



- fork the main process into multiple threads
- all threads access the same memory, i.e. no explicit memory transfers
- in simplest case only loops need to be adopted

OpenMP Example




```fortran
1  !$OMP PARALLEL DO
2  do i = 1, length
3     r(i) = a(i) + b(i)
4  end do
5  !$OMP END PARALLEL DO
```

- lines 2-4: simple FORTRAN loop to sum arrays a and b into r
- line 1: fork OpenMP threads for loop parallelisation, automatic load balancing
- line 5: join the threads

However, reality isn't that easy...

OpenMP Challenges (Examples)

Parallel Task Identification

- simple indepentent loop iterations are fine
- a compiler is not able to check for in complex loops

$\rightarrow$ programmer must check for himself

Data Races

- concurrent data write access
- neither compiler nor the hardware may prevent it

$\rightarrow$ programmer must take care for

Loop Carried Dependencies

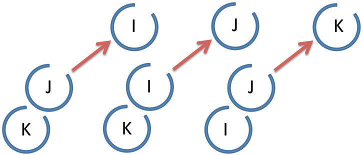
- loop iterations depend on previous interations
- only algorithmic restructure may help

$\rightarrow$ programmer must redesign algorithm

Tophat Filter – Loop Restructure

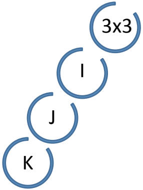
Not suitable for OpenMP:

- muliple nested loops
- function call (filter kernel) and memory copies

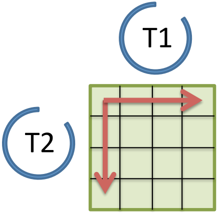


Loop restructure

- combine all loops (outer + kernel) into a single simple loop
- execute outer loop (K) in parallel



Wall Loops – Atomic Operations

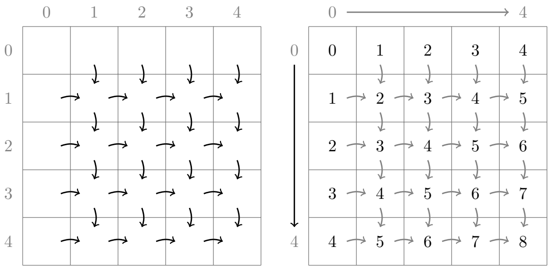


- threads access the same cells
- results in a race condition

```
1 WALL_LOOP2:
2 DO IW=1, N_CELLS
3   [...]
4   SELECT CASE(IOR)
5     CASE( 1)
6       !$OMP ATOMIC WRITE
7       RHO_DZDX(I-1,J,K) = RHO_DZDN
8       !$OMP END ATOMIC
9   [...]
```

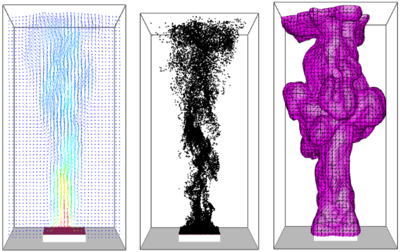
- line 6+8: instruct OpenMP to restrict concurrent access
- introduces overhead

Radiation Solver – Loop Carried Dependencies



- a simple index looping prevents parallelisation
- idea: parallel execution inside of wavefronts
- forced to restructure algorithm
- new one computes the same results

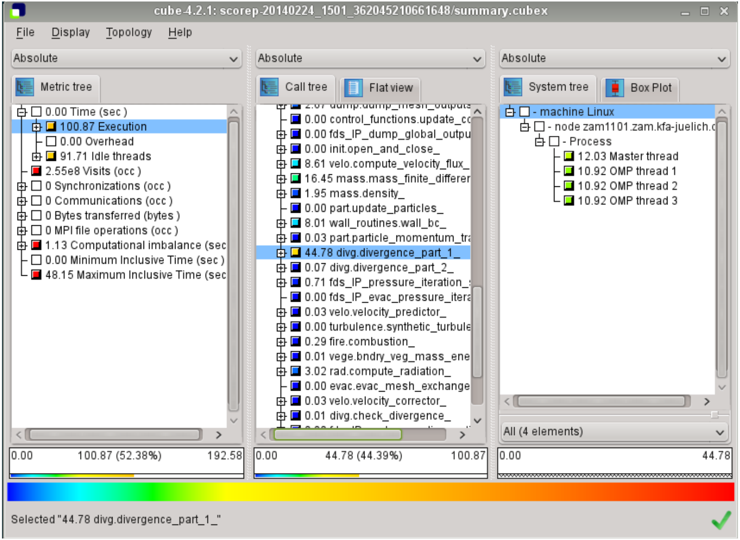
Benchmark – FDS Scenario and Computer Systems



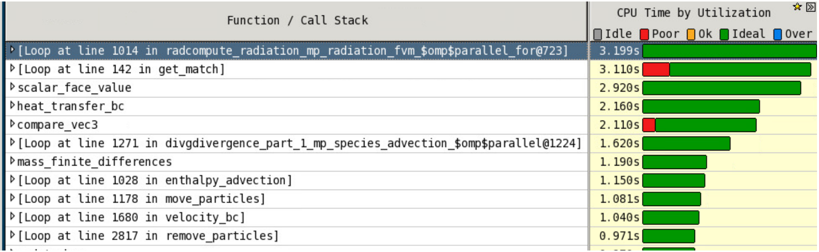
- `bench2` input shipped with FDS
- various grid sizes
- fixed number of pressure iterations

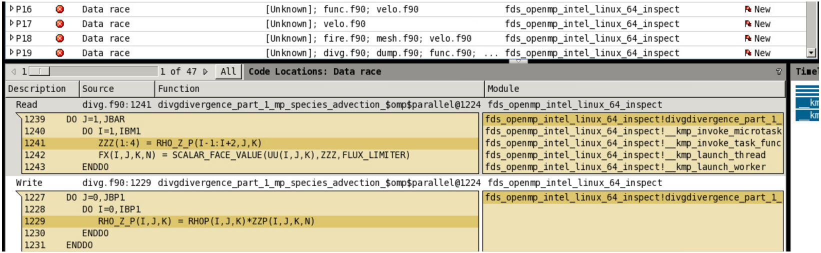
	workstation	**juropa2**	**juropa3**
processor(s)	i7-2600	2x Xeon X5570	2x Xeon E5-2650
clockspeed	3.4 GHz	2.93 GHz	2.0 GHz
cores (threads)	4 (8)	8 (16)	16 (32)
cachesize	8 MB	8 MB	20 MB
memory bandwidth	21 GB/s	32 GB/s	51.2 GB/s

Tools – Scalasca



Tools – VTune


Function / Call Stack
CPU Time by Utilization
Idle Poor Ok Ideal Over
[Loop at line 1014 in radcompute_radiation_mp_radiation_fvm_ompparallel_for@723] 3.199s
[Loop at line 142 in get_match] 3.110s
scalar_face_value 2.920s
heat_transfer_bc 2.160s
compare_vec3 2.110s
[Loop at line 1271 in divgdivergence_part_1_mp_species_advection_ompparallel@1224] 1.620s
mass_finite_differences 1.190s
[Loop at line 1028 in enthalpy_advection] 1.150s
[Loop at line 1178 in move_particles] 1.081s
[Loop at line 1680 in velocity_bc] 1.040s
[Loop at line 2817 in remove_particles] 0.971s


P16 Data race [Unknown]; func.f90; velo.f90 fds_openmp_intel_linux_64_inspect New
P17 Data race [Unknown]; velo.f90 fds_openmp_intel_linux_64_inspect New
P18 Data race [Unknown]; fire.f90; mesh.f90; velo.f90 fds_openmp_intel_linux_64_inspect New
P19 Data race [Unknown]; divg.f90; dump.f90; func.f90; ... fds_openmp_intel_linux_64_inspect New
1 of 47 All Code Locations: Data race
Description Source Function Module
Read divg.f90:1241 divgdivergence_part_1_mp_species_advection_ompparallel@1224 fds_openmp_intel_linux_64_inspect
1239 DO J=1,JBAR
1240 DO I=1,IBM1
1241 ZZZ(1:4) = RHO_Z_P(I-1:I+2,J,K)
1242 FX(I,J,K,N) = SCALAR_FACE_VALUE(UU(I,J,K),ZZZ,FLUX_LIMITER)
1243 ENDDO
fds_openmp_intel_linux_64_inspect!divgdivergence_part_1_
fds_openmp_intel_linux_64_inspect!__kmp_invoke_microtask
fds_openmp_intel_linux_64_inspect!__kmp_invoke_task_func
fds_openmp_intel_linux_64_inspect!__kmp_launch_thread
fds_openmp_intel_linux_64_inspect!__kmp_launch_worker
Write divg.f90:1229 divgdivergence_part_1_mp_species_advection_ompparallel@1224 fds_openmp_intel_linux_64_inspect
1227 DO J=0,JBP1
1228 DO I=0,IBP1
1229 RHO_Z_P(I,J,K) = RHOP(I,J,K)*ZZP(I,J,K,N)
1230 ENDDO
1231 ENDDO
fds_openmp_intel_linux_64_inspect!divgdivergence_part_1_

Serial Timing

Top-Down View

function	t[s]	t[%]
divergence_part_1	48.4	33.7
compute_velocity_flux	20.0	13.9
mass_finite_differences	15.9	11.1
compute_radiation	13.4	9.3
update_particles	7.4	5.2

Buttom-Up View

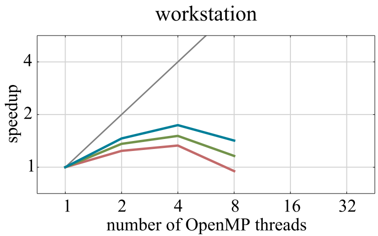
function	t[s]	t[%]
scalar_face_value	14.8	15.9
get_sensible_enthalpy_diff	4.0	4.3
loop,l.1012,radiation_fvm	3.7	4.0
loop,l.151,div._part_1	2.4	2.6
loop,l.672,velocity_flux	2.4	2.6

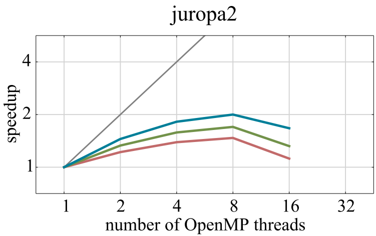
Parallelised Routines

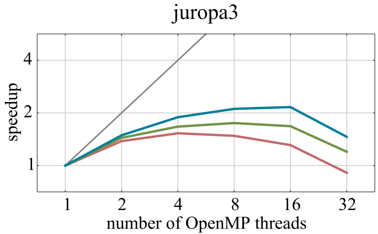
	Runtimes s		
Function	Serial	OpenMP	Parallel Percentage
divergence_part_1	37.4	16.8	82.8
species_advection	9.7	4.6	78.4
radiation_fvm	9.6	7.3	87.5
compute_viscosity	7.3	3.4	79.4
enthalpy_advection	7.0	3.5	69.5
mass_finite_differences	7.0	3.9	75.7
velocity_flux	6.7	3.2	97.2
density_advection	4.8	2.4	65.4
pressure_solver	4.0	5.0	17.2
test_filter	2.0	0.6	99.4
baroclinic_correction	1.8	1.1	99.8
openmp_check	0.0	0.0	6.0

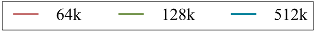
- all changed routines are working fine
- an adequate parallelisation level in these routines was reached

OpenMP Scaling







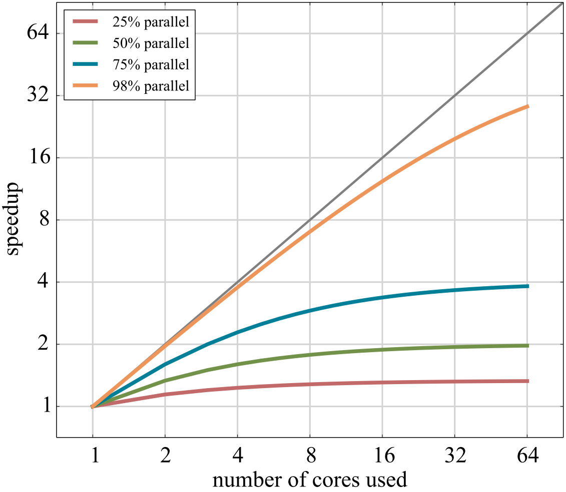


- speedup increases with larger grids
- hyperthreading is contra-productive
- maximal speedup of two

Overall Parallelisation

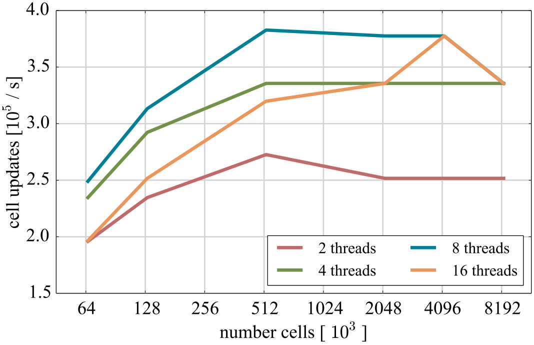
	Serial		OpenMP (4 threads)	
Function	s	%	s	Parallel Percentage
MAIN	94.7	100.0	66.5	39.7
DIVG	29.5	31.2	15.4	19.9
MASS	9.9	10.4	6.7	4.5
VELO	16.6	17.5	9.5	9.9
PRES	4.0	4.2	5.0	0.0
WALL	3.9	4.1	4.0	0.0
DUMP	6.2	6.6	7.6	0.0
PART	6.8	7.2	6.9	0.0
RADI	12.7	13.4	6.2	9.2
FIRE	2.1	2.2	2.1	0.0
COMM	0.0	0.0	0.0	0.0

Amdahl's Law



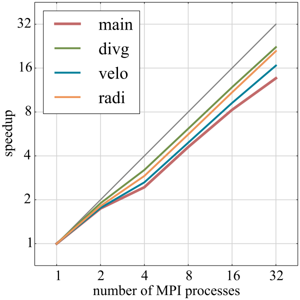
25% parallel
50% parallel
75% parallel
98% parallel
speedup
64
32
16
8
4
2
1
1
2
4
8
16
32
64
number of cores used

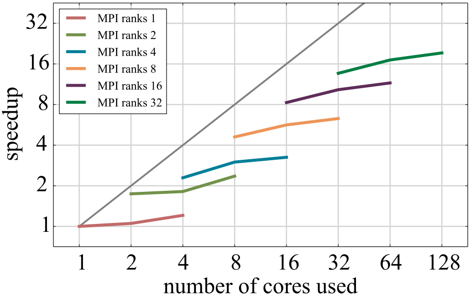
OpenMP Performance



- cell updates increase with number of threads used
- the performance stagnates above the 512k grids
- memory access performance as potential bottleneck

Hybrid Scaling





- MPI offers much greater speedup, w.r.t. the pure OpenMP version
- hybrid (MPI and OpenMP) use is possible

Summary

- easy to use: just by setting `OMP_NUM_THREADS` and if necessary `OMP_STACKSIZE` to achieve a speedup of two on four cores
- difficult to programm: many pitfalls are only trackable with tools like VTune, algorithmic restructure needed
- MPI outperforms OpenMP

- in general: the achieved performance / scaling is bad
- in the FDS context: it is fine