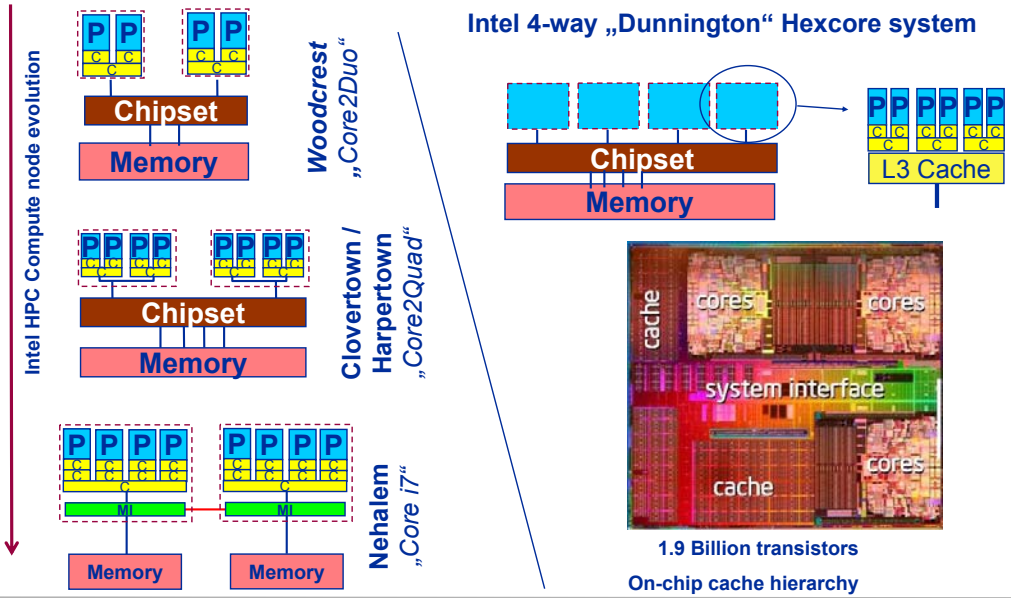
Multicore-aware wavefront parallelization of a lattice Boltzmann flow solver

J. Habich, Thomas Zeiser, G. Hager, G. Wellein
Erlangen Regional Computing Center (RRZE)
Friedrich-Alexander-University Erlangen-Nuremberg
hpc@rrze.uni-erlangen.de

This work is financially supported by:
KONWIHR II
Federal Ministry of Education and Research
through grant SKALB (01IH08003)

The x86 multicore evolution
Benchmark architectures: Intel Quad-/Hexa-Cores



The x86 multicore evolution
Benchmark architectures: Intel Quad-/Hexa-Cores

Type		Woodcrest Xeon 5160 @3.0 GHz	Dunnington Xeon 7460 @2.66 GHz	Nehalem Xeon @2.66 GHz
L1 group	size [kB]	32	32	32
	TRIAD GB/s	3.7	3.0	11.6
L2 group	L2 size [MB]	4	3	0.25
	shared by # cores	2	2	1
	TRIAD GB/s	3.7	3.5	see L1
L3 group / socket	L3 size [MB]	-	16	8
	shared by # cores	-	6	4
	TRIAD GB/s	-	3.5	16.6
System	# sockets	2	4	2
	raw bw [GB/s]	21.3	34.0	51.2
	TRIAD GB/s	6.7	13.2	32.7

Stream TRIAD (array size: 20,000,000; `nontemporal stores` via compiler)

Nehalem: Early access; pre-production; SMT disabled

Lattice Boltzmann method
A simple but efficient kernel F(x,y,z,0:18,t)

```
double precision F(0:xMax+1,0:yMax+1,0:zMax+1,0:18,0:1)
do z=1,zMax
   do y=1,yMax
      do x=1,xMax
         if( fluidcell(x,y,z) ) then
            LOAD F(x,y,z, 0:18,t)
            Relaxation (complex computations)
            SAVE F(x  ,y  ,z  , 0,t+1)
            SAVE F(x+1,y+1,z  , 1,t+1)
            SAVE F(x  ,y+1,z  , 2,t+1)
            SAVE F(x-1,y+1,z  , 3,t+1)
            ...
            SAVE F(x  ,y-1,z-1,18,t+1)
         endif
      enddo
   enddo
enddo
```

2*19 caches lines are touched for a single cell update – however they are accessed contiguously

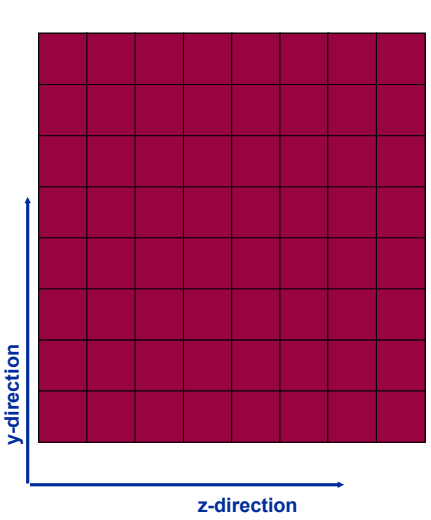
High spatial data locality if 38 cache lines stay in the cache!

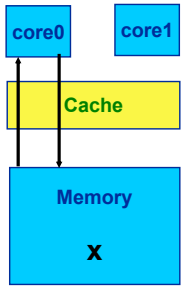
(38 * 128 Byte ~ 5 kByte << L2/L3 caches)

Lattice Boltzmann method
A simple performance model for the standard implementation

- Standard performance measure for LBM:
  Million (fluid) node updates per second: MLUPS/s
- MFLOP/s is not a good idea: 150 – 400 FLOP for kernel:
  - implementation
  - compiler version & compiler options
  - divide?!
- Estimate maximum performance on basis of attainable main memory bandwidth (MBW):
  - Data transfers / LUP = 3*19*8Byte = 456 Byte
  - Performance estimate [MLUPS/s]: MBW [MByte/s] / 456 Byte
  - MBW is determined through low level kernel, e.g. STREAM

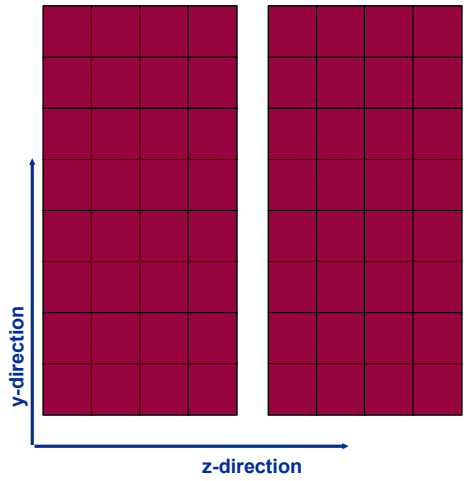
LBM & multicore architectures
Standard sequential implementation

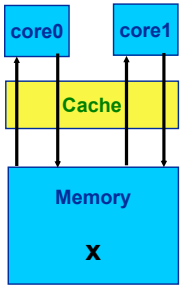





```
do z=1,zMax
   do y=1,yMax
      do x=1,xMax
           if( fluidcell(x,y,z) ) then
           endif
      enddo
   enddo
enddo
```

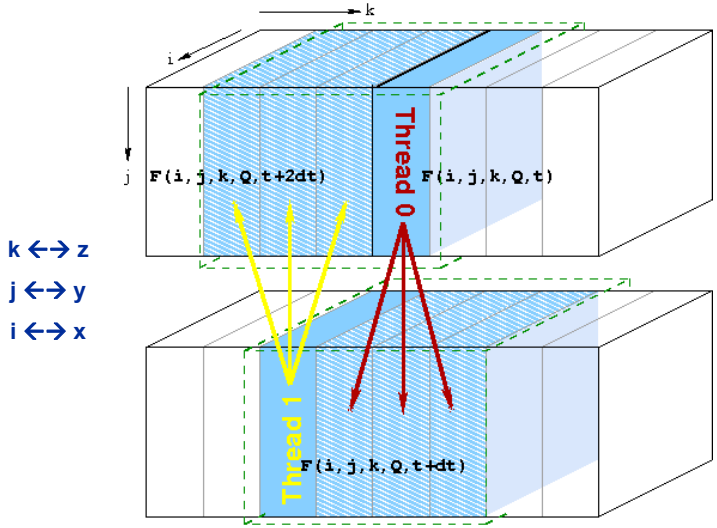

LBM & multicore architectures
Standard naive shared memory parallel






```
!$OMP PARALLEL DO private(…)
do z=1,zMax
   do y=1,yMax
      do x=1,xMax
           if( fluidcell(x,y,z) ) then
           endif
      enddo
   enddo
enddo
```


LBM & mutlicore architectures
A different parallel approach: Propagating wavefronts!

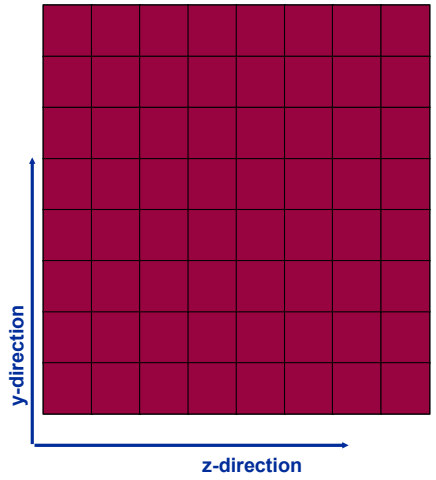


k ↔ z
j ↔ y
i ↔ x

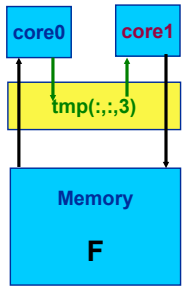
core0: $F(:,:,z-1:z+1,0:18,1)_t$ → *tmp(:,:,0:18mod(k,4))*

core1: *tmp(:,:,0:18,mod(z-3,4):mod(z-1,4))* → $F(:,:,z-2,0:18,1)_{t+2}$

LBM & mutlicore architectures
Propagating two wavefronts!



`F(:,:,:,:,1)` is obsolete!



Use small buffer
`tmp(:,:,0:18, 0:3)`
which fits into the cache

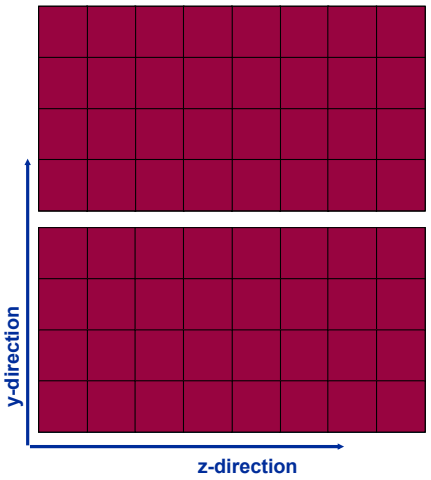
Save main memory data transfers for `F(,1)` !

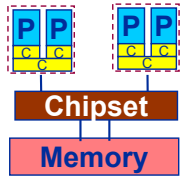
Sync threads/cores after each z-iteration

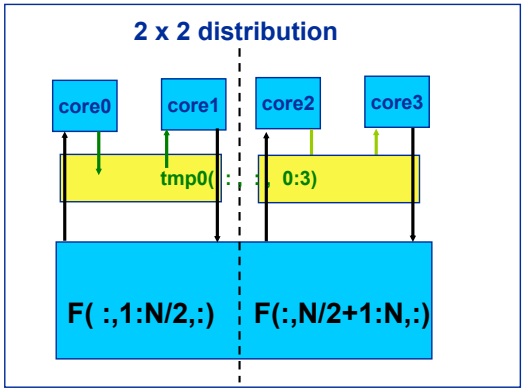
core0: $F(:,:,z-1:z+1,0:18,0)_t$ → *tmp(:,:,0:18,mod(k,4))*

core1: *tmp(:,:,0:18,mod(z-3,4):mod(z-1,4))* → $F(:,:,z-2,0:18,0)_{t+2}$

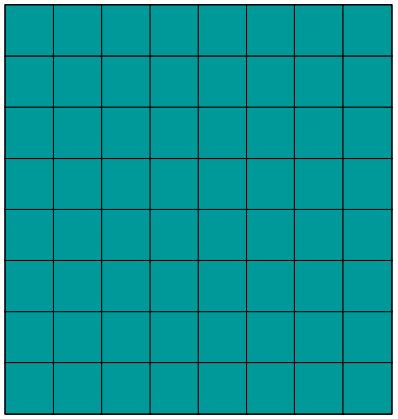
LBM & multicore architectures
Propagating two groups of wavefronts (2x2)







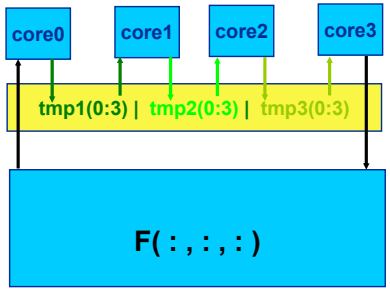
LBM & multicore architectures
Propagating four wavefronts on native quadcores (1x4)



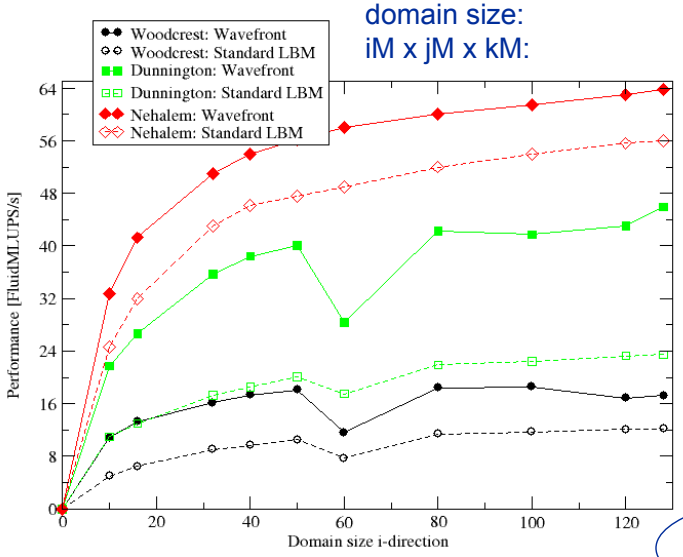
Running `tb` wavefronts requires `tb-1` temporary arrays `tmp` to be held in cache!

Extensive use of cache bandwidth!

1 x 4 distribution



LBM & multicore architectures
Full node results with 2 wavefronts



domain size:
iM x jM x kM:

kM=600

jM = 32 (Woodcrest),

jM = 48 (Dunnington),

jM = 64 (Nehalem)

Cache size needs to be larger than:

8 * 19 * iM*jM* 8 Bytes

(= 4.7 MB for iM=120; jM=32)

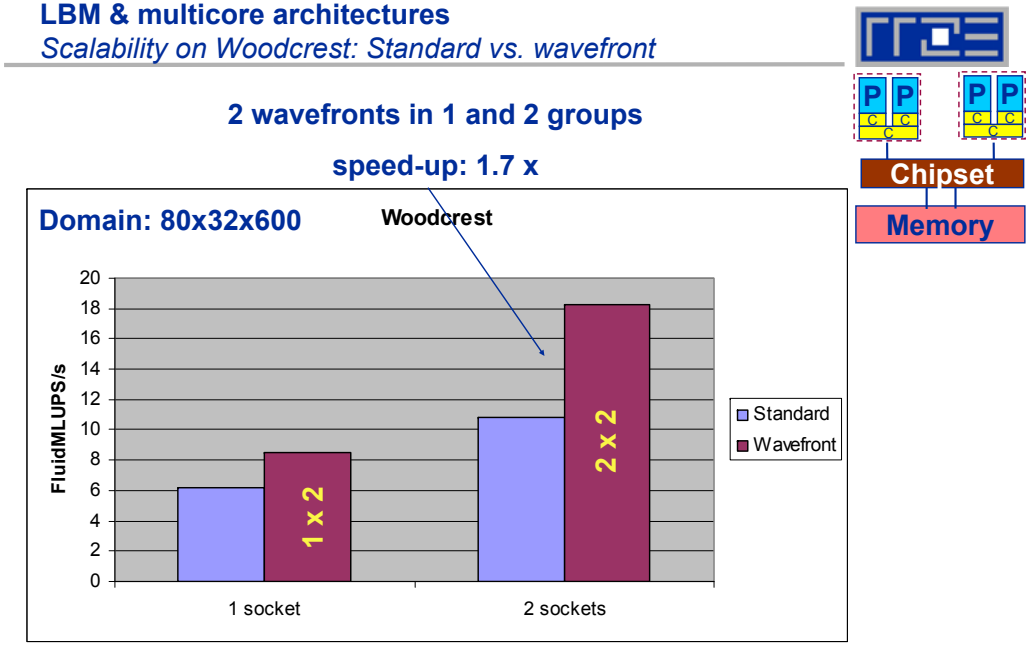
spatial blocking in j-direction required!

LBM & multicore architectures

Scalability on Woodcrest: Standard vs. wavefront

2 wavefronts in 1 and 2 groups

speed-up: 1.7 x

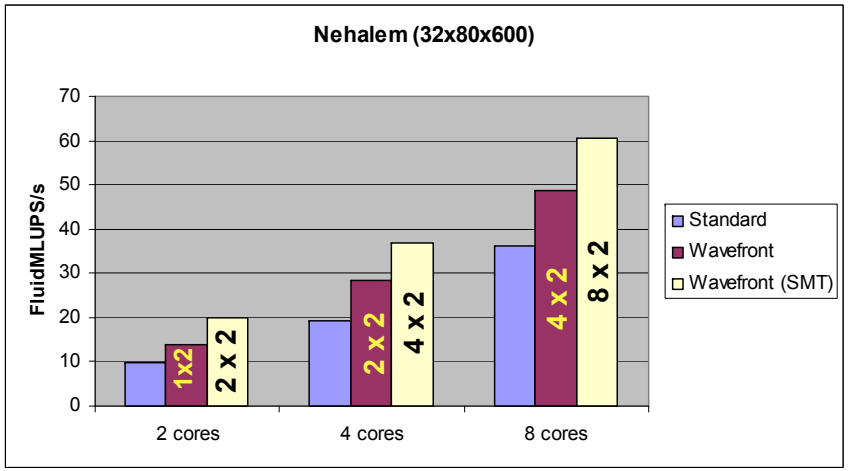


LBM & multicore architectures

Scalability on Nehalem: Standard vs. wavefront

2 wavefronts with 1, 2 and 4 groups

SMT with 2,4 and 8 groups helps a lot!



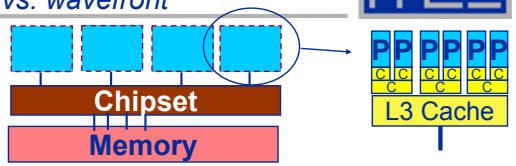
LBM & multicore architectures

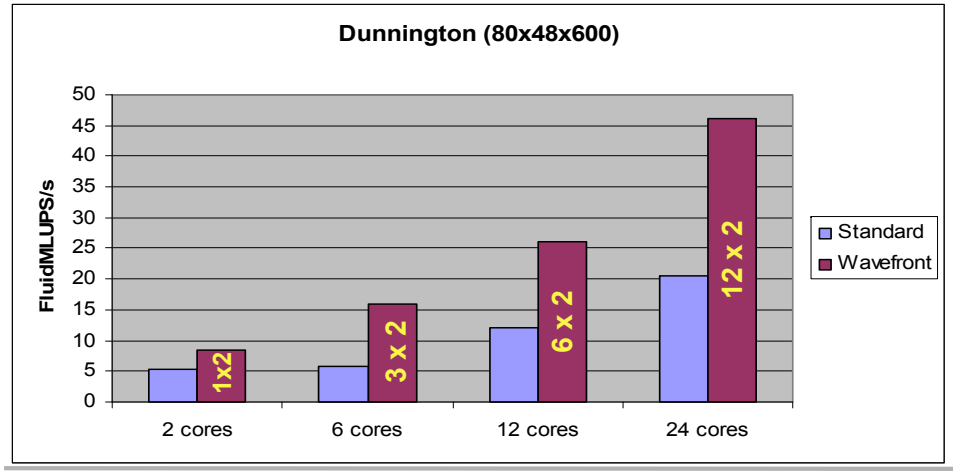
Scalability on Dunnington: Standard vs. wavefront

2 wavefronts with

1, 3, 6 and 12 domains

2.2x speed-up





LBM & wavefront approach: Status & Outlook

- **to do:**
  - blocking in j-/y-direction in progress
  - no in-cache optimizations implemented / explored so far
  - perfomance model
- **wavefront parallelization for LBM particularly beneficial if**
  - multi-core chip is bandwidth starved (one core can sustain main memory bandwidth)
  - Large shared (on-chip) cache is available
- **can easily be implemented for other stencil based methods:**
  - Jacobi solver/smoother → COMPSAC2009
  - Gauß-Seidel → first tests
- **Easy to implement and parallelize but hybrid MPI/OpenMP approach is required if used in a massively parallel production code.**