

Large-Scale Jet Simulations on a Vector Supercomputer

Volker Gaibler (LSW Heidelberg)

Max Camenzind (LSW)

Martin Krause (MPE Garching)

JETSET School Galway

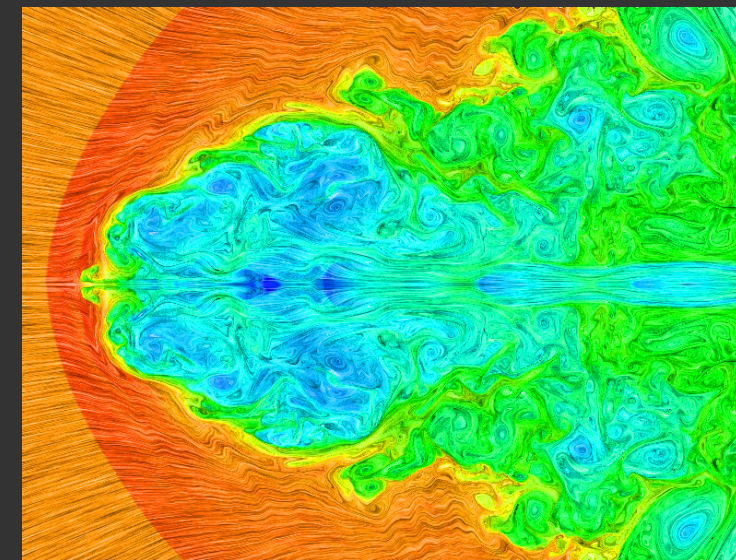
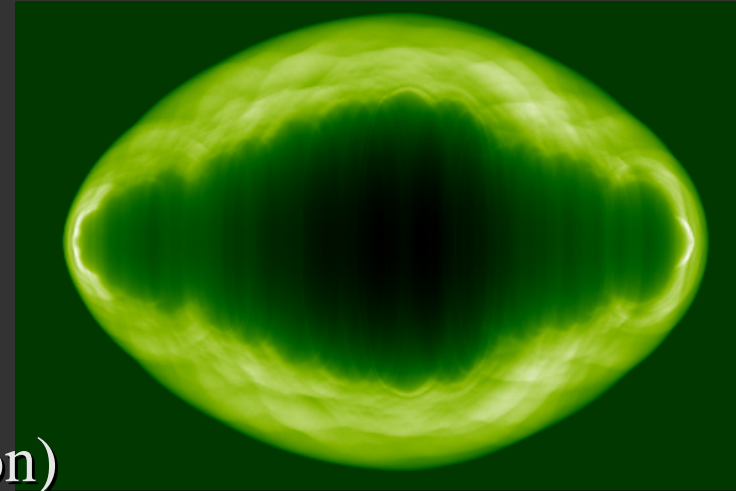
January 2008

Outline

- Scientific problem
 - Computational demands
 - Concept of vectorization
 - Our domain decomposition
 - Vectorization
 - Shared-memory parallelization
 - MPI parallelization
-
-

Science

- Extragalactic jets in galaxy clusters
 - other difficulties than for YSO jets
 - very low jet density
 - slow jet head propagation
 - interaction in midplane (bipolar simulation)
 - roundish bow shocks
 - turbulent wide cocoons
 - MHD



Computational Demands

- MHD in large computational domain
4000 x 1600 cells in 2.5D (axisymmetry)
 - Slow jet head and realistic jet lengths:
need several millions of timesteps
 - Unfortunately 'time-coordinate' not parallelizable...
-
-

NEC SX-8

- Vector supercomputer at HLRS Stuttgart
- Hybrid system:
 - 72 nodes
 - 8 vector CPUs per node,
128 GB shared memory
 - nodes with IXS network, 16 GB/s bidirectional per node
(576 CPUs in total)
 - Cluster filesystem

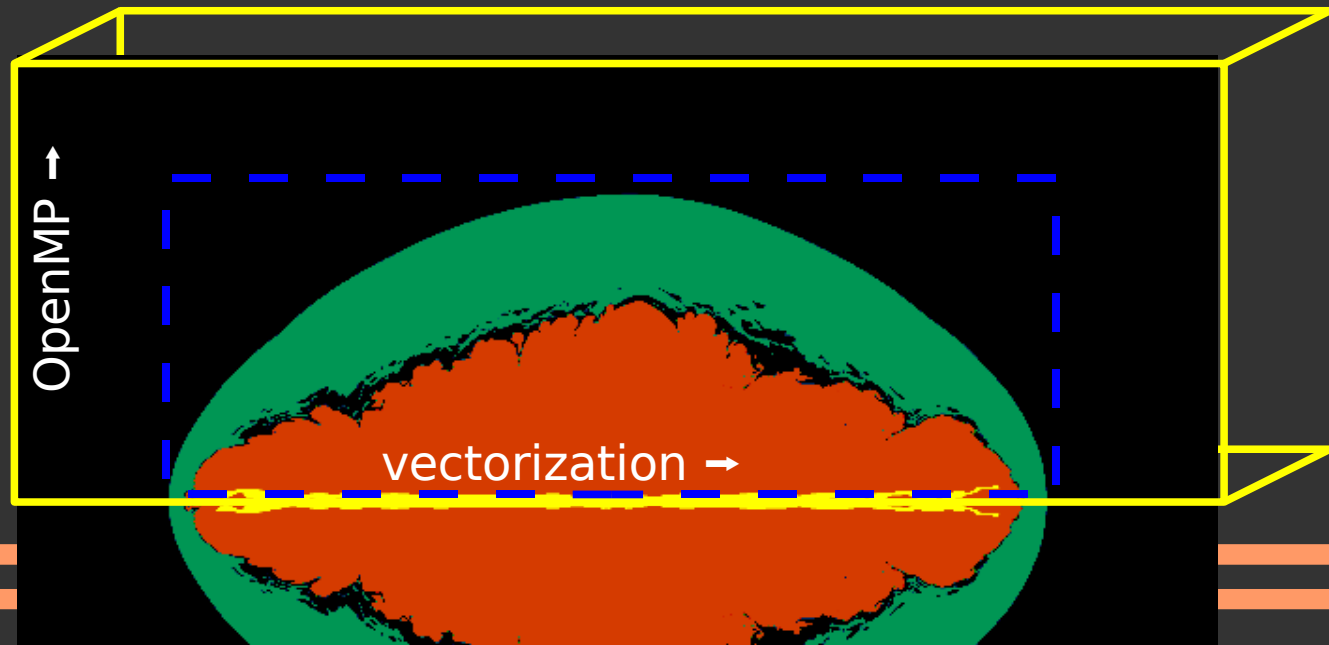


Vector Computers

- Scientific computing: usually computations on huge amount of data
 - Pipelining:
 - *instruction* pipelining in PCs
 - additionally *data* pipelining in vector machines (SIMD)
 - Need fast memory access:
 - 4096 memory banks per node
 - gives sustained performance of 64 GB/s for each CPU to full RAM, only stride 4, 8, ... penalties
 - Think of 'computing with vectors' – easy to implement!
 - need long loops for maximum performance!
 - avoid conditionals in innermost loop
-
-

Domain Decomposition

- How to combine the different concepts?
Directions treated differently:
 - Vectorization for direction with most cells
 - Shared-memory parallelization for second direction
 - Smart domain handling: ignore matter outside bow shock
- MPI parallelization for second and third direction
(M. Krause)



Example Code

```
for (iz=0; iz<=pny; iz++) {  
    #pragma pdir parloop  
    for (iy=0; iy<=pny; iy++) { ←————— Microtasking (OpenMP-like)  
        #pragma vdir nodep  
        for(ix=0; ix<=pnx; ix++) { ←————— vectorization  
            w5[i][iz][iy][ix] = g->rhoi[i][iz][iy][ix] / g->rho[iz][iy][ix];  
            ...  
        }  
    }  
}
```

MPI Parallelization

- Used for second and third direction
- Good performance for 3D
- Not very efficient for 2D:
 - too many processors for the problem (comms!)
 - better stick with shared-memory

Summary

- Vectorization is a simple way to gain great performance
- Generally easy to implement
 - just adding compiler pragmas
 - try to avoid conditionals in innermost (long) loops
 - may be difficult with AMR or unstructured grids
 - architecture-specific