OpenMP parallelization of the complex magnetohydrodynamic model BATS-R-US

Gábor Tóth
Hongyang Zhou

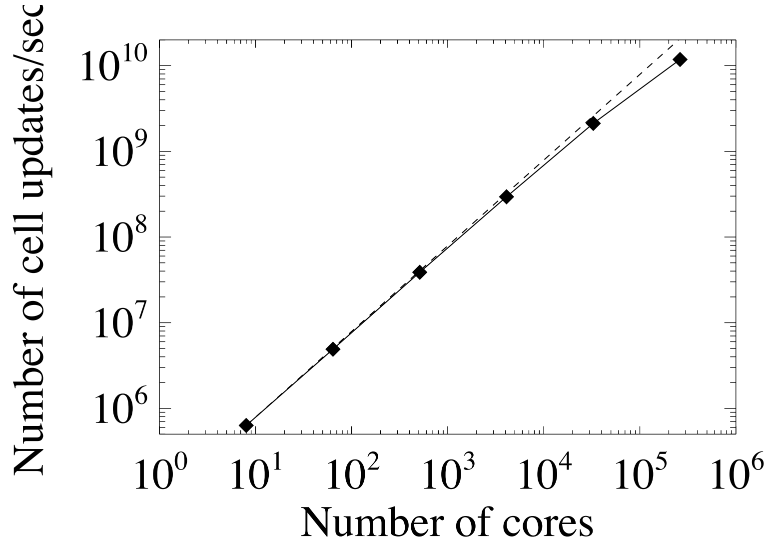
Department of Climate and Space
Center for Space Environment Modeling
University Of Michigan

BATS-R-US

Physics

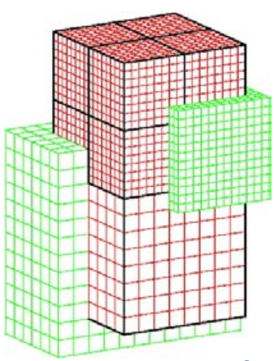
- Classical, semi-relativistic and Hall MHD
- Multi-species, multi-fluid, 5 and 6-moment
- Anisotropic pressure for ions and electrons
- Radiation hydrodynamics multigroup diffusion
- Multi-material, non-ideal equation of state
- Heat conduction, viscosity, resistivity
- Alfven wave turbulence and heating

Parallel scaling from 8 to 262,144 cores on Cray Jaguar. 40,960 grid cells per core in 10 grid blocks with 16x16x16 cells.



Numerics

- Parallel Block-Adaptive Tree Library (BATL)
- Cartesian and generalized coordinates
- Splitting the magnetic field into $B_0 + B_1$
- Divergence B control: 8-wave, CT, projection, parabolic/hyperbolic
- Numerical fluxes: Godunov, Rusanov, AW, HLLE, HLLC, HLLD, Roe, DW
- Explicit, local time stepping, limited time step, sub-cycling
- Point-, semi-, part and fully implicit time stepping
- Up to 4th order accurate in time and 5th order in space



Applications

- Heliosphere, sun, planets, moons, comets, HEDP experiments

250,000+ lines of Fortran 90+ code with MPI parallelization

Challenges

- **Why OpenMP?**
  - Using pure MPI, replicated data structures (like block tree, large lookup tables…) cannot fit in memory for very large grid
  - OpenMP reduces the memory use by using fewer MPI processes, while maintaining speed via multithreading
  - Allows the use of smaller blocks and/or scaling to larger number of cores
- **Hybrid Parallelization Options**
  - Multi-threading for grid cells: fine-grained
    - Many loops to be parallelized
    - Significant work is done outside these loops
  - Multi-threading for grid blocks: coarse-grained
    - Fewer loops to be parallelized
    - Most of the work is multi-threaded
    - Many variables need to be declared thread-private: module variables, saved variables, initialized variables
    - Race conditions are very difficult to debug: Intel INSPECTOR

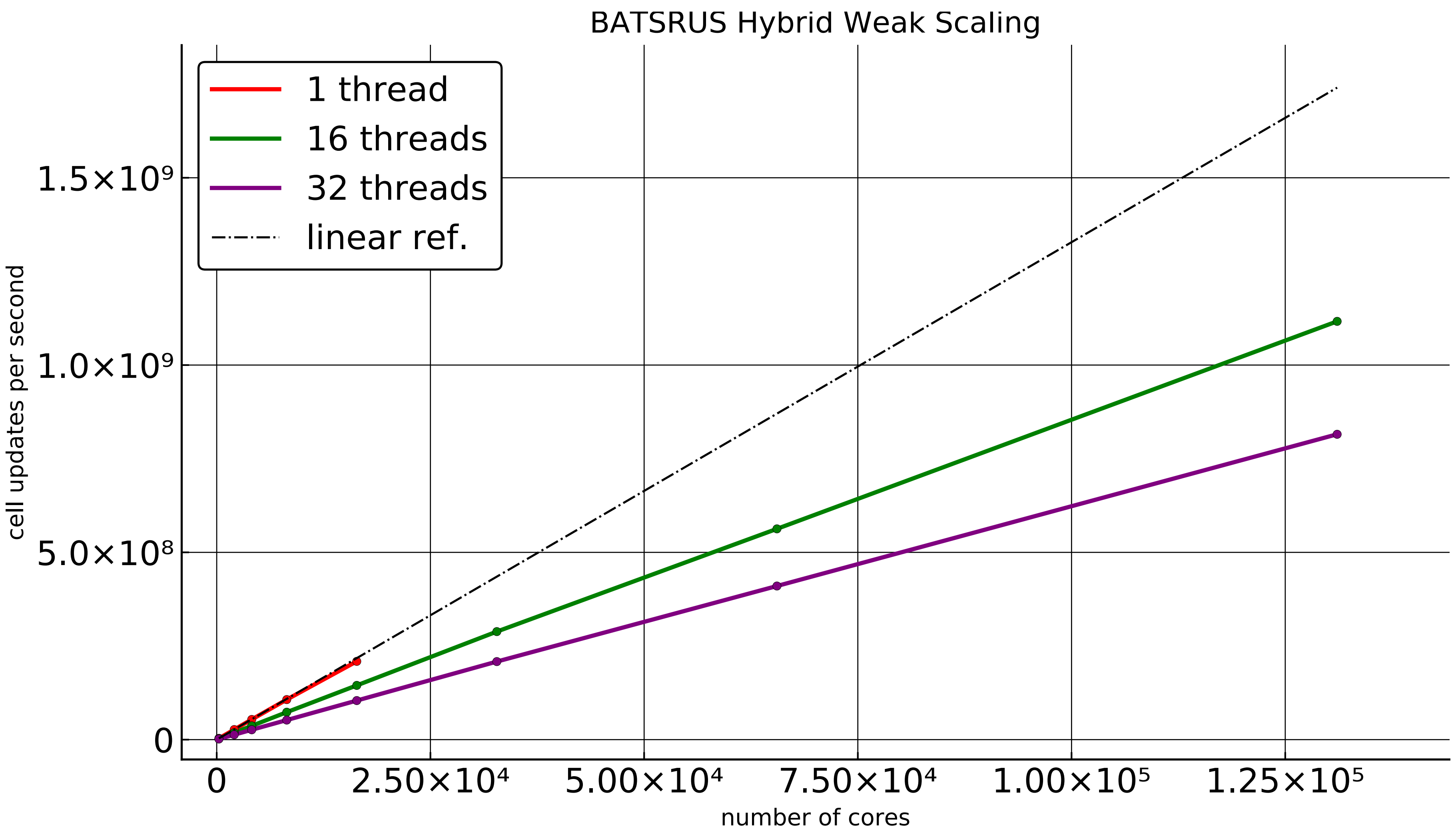
Variable declarations and allocations

```
! Primitive variables extrapolated from left and right
 real, allocatable:: LeftState_VX(:,:,:,:), RightState_VX(:,:,:,:)
 real, allocatable:: LeftState_VY(:,:,:,:), RightState_VY(:,:,:,:)
 real, allocatable:: LeftState_VZ(:,:,:,:), RightState_VZ(:,:,:,:)
 !$omp threadprivate( LeftState_VX, RightState_VX )
 !$omp threadprivate( LeftState_VY, RightState_VY )
 !$omp threadprivate( LeftState_VZ, RightState_VZ )
…
 !$omp parallel
 allocate(LeftState_VX(nVar,nI+1,nJ,nK), RightState_VX(nVar,nI+1,nJ,nK))
 allocate(LeftState_VY(nVar,nI,nJ+1,nK), RightState_VY(nVar,nI,nJ+1,nK))
 allocate(LeftState_VZ(nVar,nI,nJ,nK+1), RightState_VZ(nVar,nI,nJ,nK+1))
…
!$omp end parallel
```

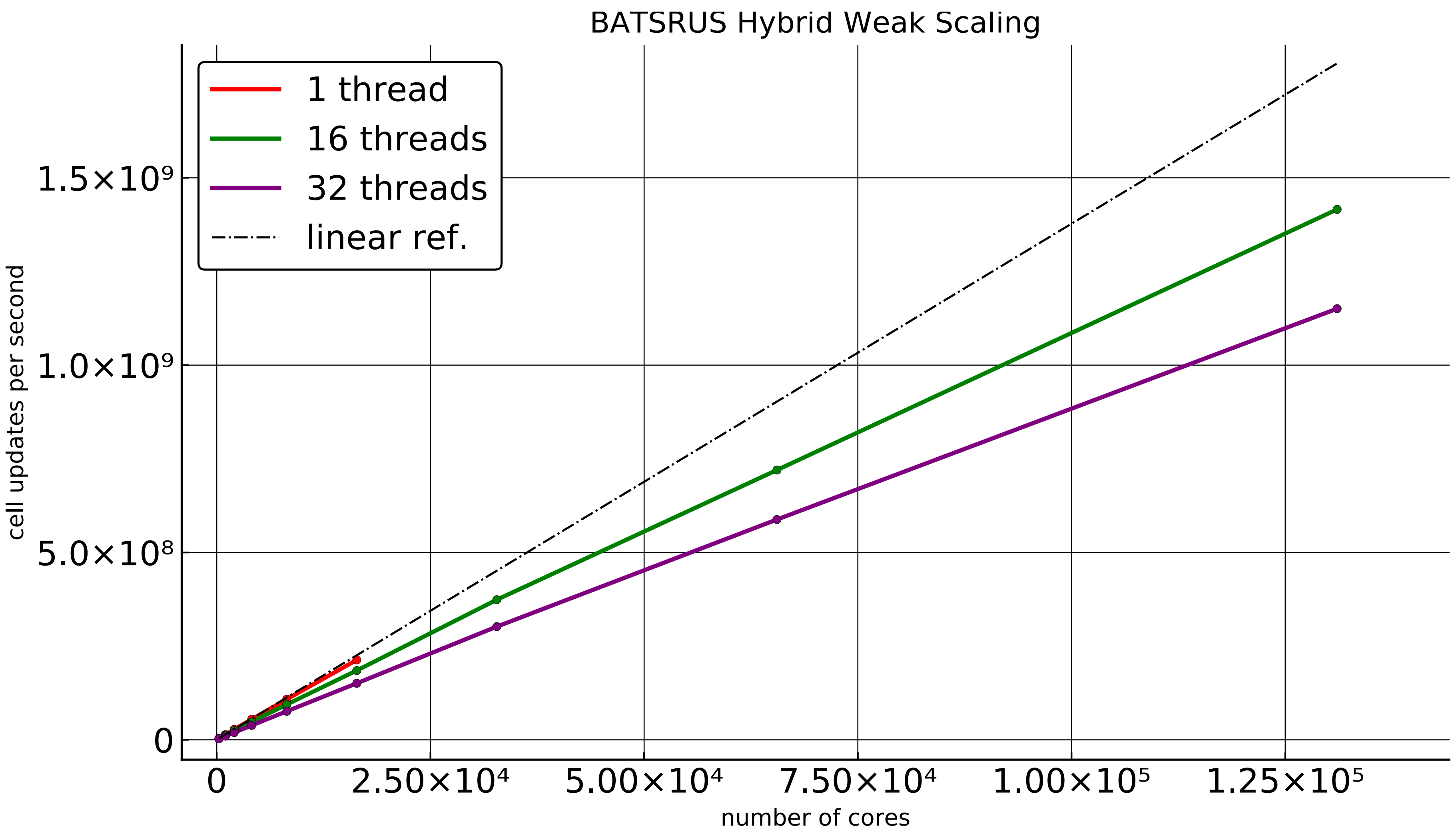
Main Loop in Explicit Solver

```
STAGELOOP: do iStage = 1, nStage
  ! Multi-block solution update.
  !$omp parallel do
  do iBlock = 1, nBlock
    if(Unused_B(iBlock)) CYCLE
    call calc_face_value(iBlock)
    call calc_face_flux(iBlock)
    call calc_source(iBlock)
    call update_state(iBlock)
    if(iStage==nStage) call calc_timestep(iBlock)
  end do
  !$omp end parallel do
  call exchange_messages
end do STAGELOOP
```

Message passing: serial



Message passing: partially multithreaded



Typical Loop in Implicit Solver

```
n = 0
do iBlock=1,nBlock

   do k=1,nK; do j=1,nJ; do i=1,nI; do iVar=1,nVar
      n = n + 1
      ! Set RHS vector
      Rhs_I(n) = Res_VCB(iVar,i,j,k,iBlock)*Dt
   end do; enddo; enddo; enddo
end do
```

Typical Loop in Implicit Solver

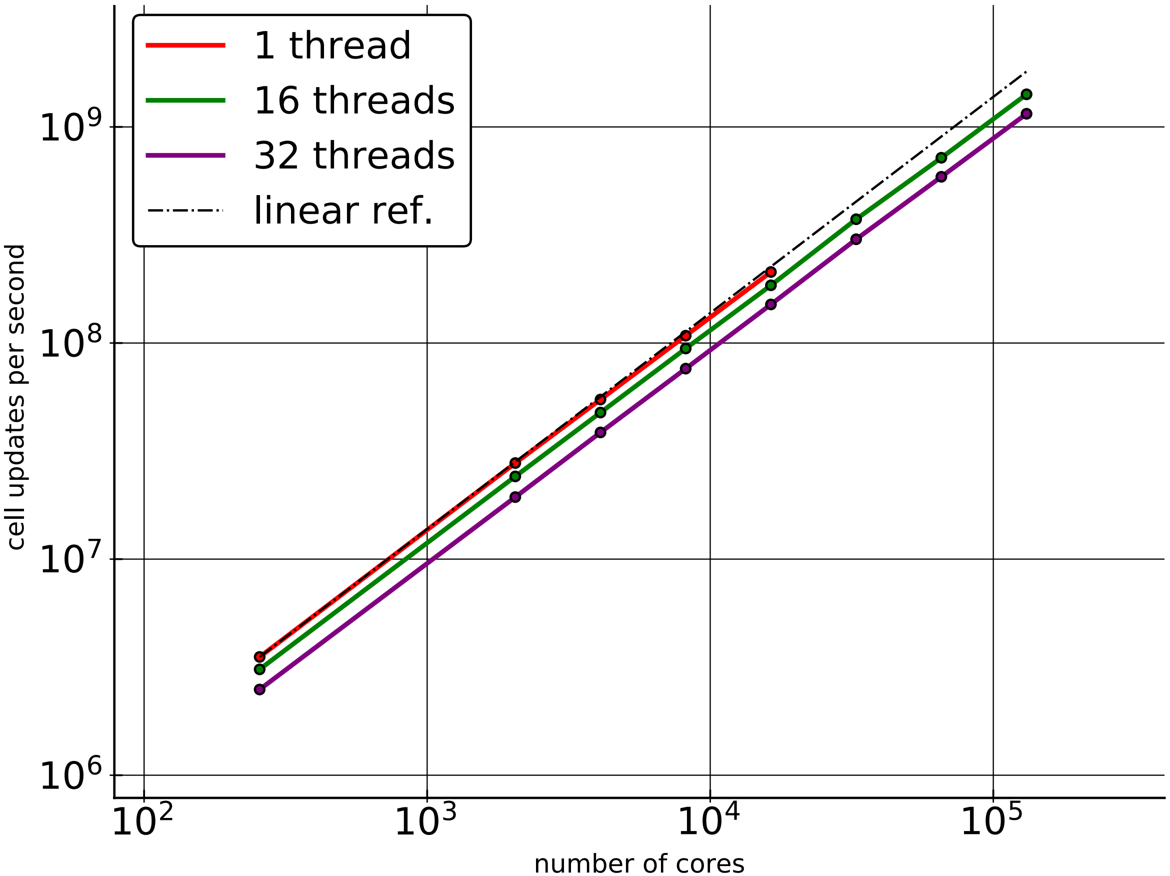
```
!$omp parallel do private( n )
do iBlock=1,nBlock
   n = (iBlock-1)*nI*nJ*nK*nVar
   do k=1,nK; do j=1,nJ; do i=1,nI; do iVar=1,nVar
      n = n + 1
      ! Set RHS vector
      Rhs_I(n) = Res_VCB(iVar,i,j,k,iBlock)*Dt
   end do; enddo; enddo; enddo
end do
!$omp end parallel do
```

Lessons Learned

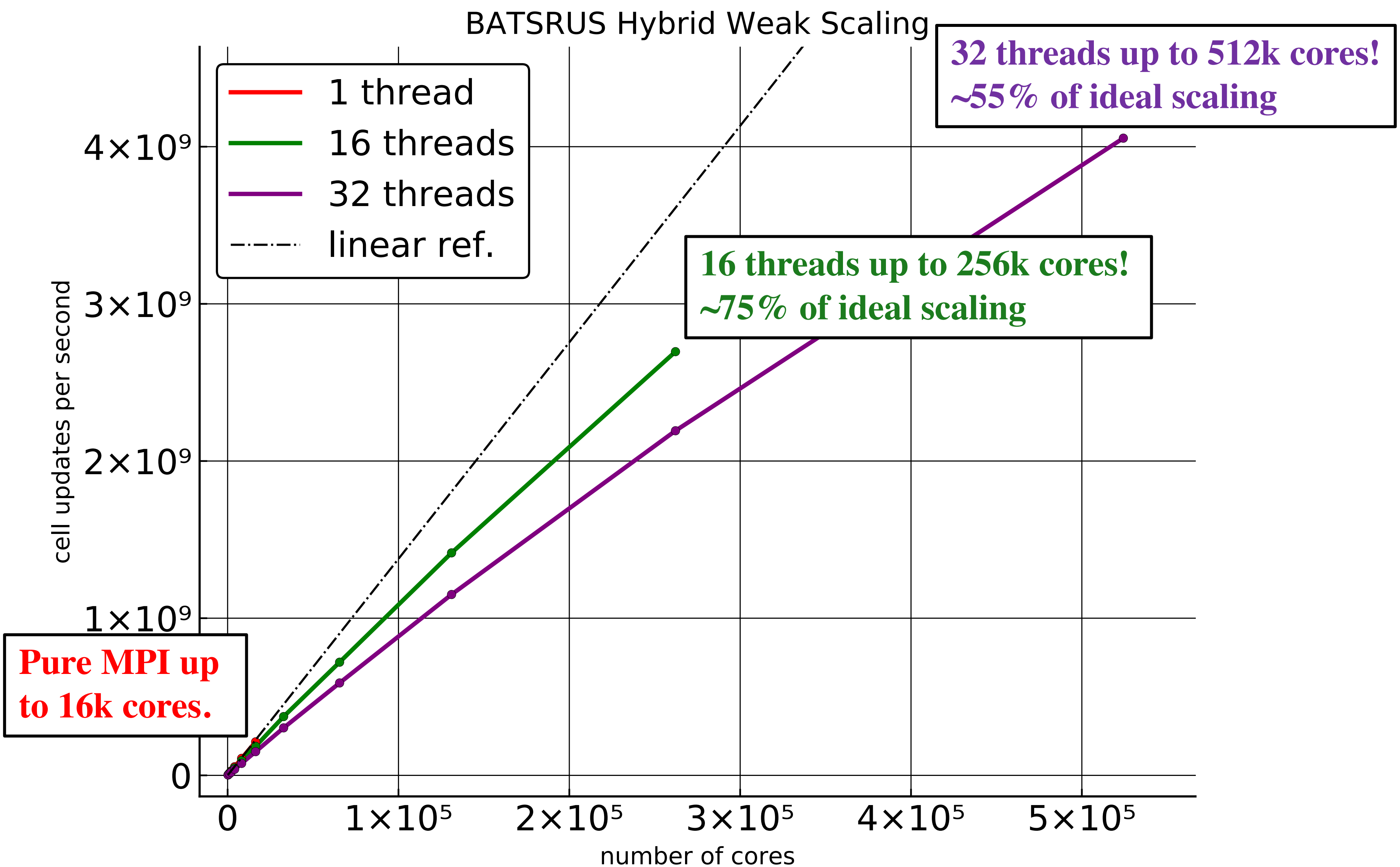
- **Code changes were surprisingly minimal**
  - 609 OpenMP directive lines (mostly thread-private declarations) were added to the 246,728 lines of source code: 0.25% change
- **Most of the time is spent on testing and debugging**
  - Comprehensive BATS-R-US nightly test suite switched to use OpenMP
  - Intel INSPECTOR was found to be the only tool to identify race conditions
  - Profiling and scaling studies revealed bottle-necks
- **Serial performance can be severely affected if code is compiled with OpenMP**
  - NAGFOR is 10 times, pgfortran 3 times, ifort 2 times slower than without OpenMP
  - gfortran and Cray fortran are not affected significantly
- **Pinning OpenMP and MPI processes on nodes is non-trivial**
  - Settings change from platform to platform, from compiler to compiler, even from one version to another version of the same compiler!
  - Instructions on web pages are often incomplete or obsolete
  - Check what actually happens with a dedicated C++ code: coreAffinity.cpp

Weak scaling on a log-log plot: explicit scheme

- **Parallel scaling and maximum problem size**
  - MHD problem on 3D uniform grid: 256 blocks with 8x8x8 cells = 131k cells per core
  - Gfortran, with optimization, +OpenMP and MPI
  - Blue Waters: 32 AMD cores per node on 2 processors, 2GB/core memory

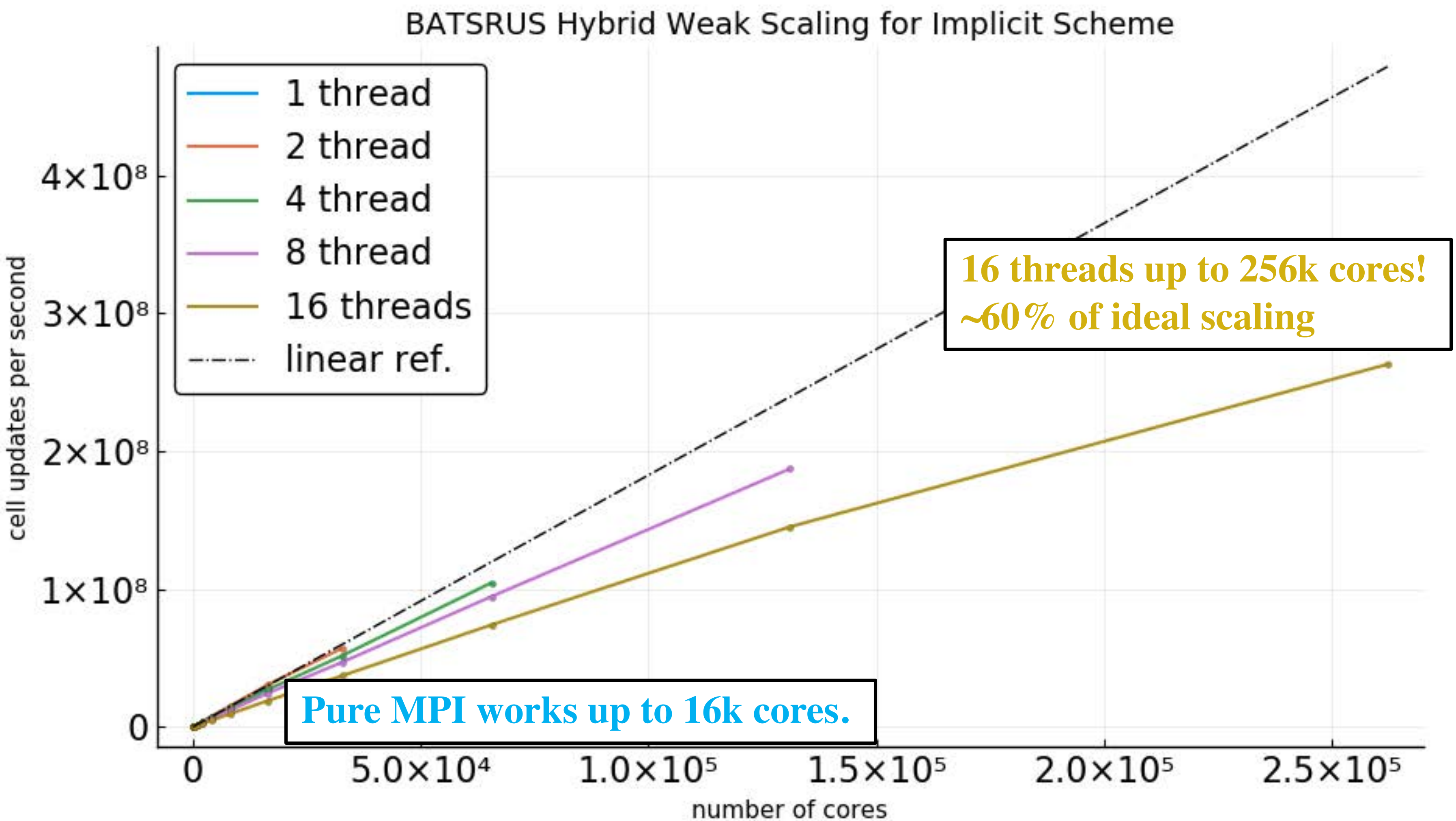


Weak scaling on a linear plot: explicit scheme



Weak scaling on a linear plot: implicit scheme

BiCGSTAB (uses less memory than GMRES)
with fixed 20 iterations per time step



Why Blue Waters?

- **Hardware**
  - Large number of cores on a uniform machine allows studying the code behavior and scaling for very large problems and finding issues like integer overflow
  - Large number of cores per node allows investigating scaling with number of OpenMP threads
- **Software**
  - Variety of compilers for testing allows identifying compiler specific issues
  - Apprentice2 / CPMAT performance tool is easy to use and useful
- **Environment**
  - Wait time for large jobs is reasonably short, so scaling studies can be done efficiently

Summary

- **We have succeeded in adding OpenMP parallelization to BATS-R-US**
  - Coarse-grain parallelization: multi-threading per grid-block
  - Relatively few changes in source code: 0.25%
    - Testing and debugging takes most time
    - A few man-month work for changing 250k lines of source code
  - Maximum problem size achievable is 32 times larger
  - Weak scaling performance is satisfactory
    - Up to 512k cores with explicit scheme: 55% of ideal scaling
    - Up to 256k cores with implicit scheme: 60% of ideal scaling
  - Compiler and platform specific issues
    - Some compilers run much slower with OpenMP
    - Pinning threads is non-trivial
- **Future work**
  - Running models with and without OpenMP together in the Space Weather Modeling Framework
  - Using GPUs…