Accelerating Computations of Turbulence on Blue Waters (at/beyond the Petascale)

P. K. Yeung (Georgia Tech, PI)
D. Pekurovsky (SDSC-UCSD)
pk.yeung@ae.gatech.edu

NEIS-P2 Blue Waters Symposium
NCSA, Urbana-Champaign, May 2013

Thanks...

NSF: PRAC and Fluid Dynamics Programs
BW & Cray staff: G. Bauer, T. Cortese, R. Fiedler, J. Kim, J. Li
D. Buaria, K. Iyer (GT); A. Majumdar (SDSC-UCSD)

Overview

- Turbulence: in short, why do we need Blue Waters
- Numerical methods and domain decomposition
- Performance factors and code development approach
- Subaward: new programming models for improvement
  - MPI-OpenMP (multithreading)
  - Co-Array Fortran (remote memory addressing)

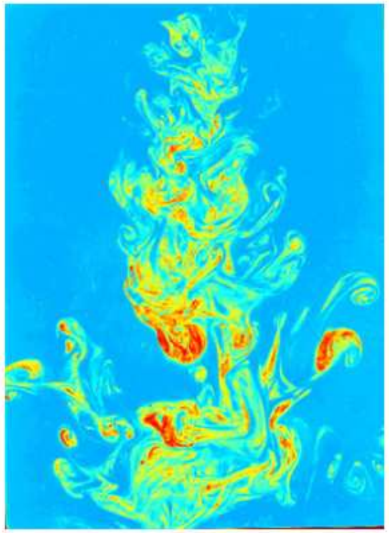
  Q: Can we overlap computation with communication?
  Discussion of performance data and current prospects
- Large-volume I/O and data archival
- Concluding remarks

Why Study Turbulence?

- It is everywhere, studied in many disciplines

Space Shuttle

Grid turbulence


Jet

- It is challenging: unsteady, 3D, stochastic, wide range of scales
- It holds the key to improved engineering devices and prediction/management of natural phenomena

Numerical Approach

- Navier-Stokes: conservation of mass ($\nabla \cdot \mathbf{u} = 0$) and momentum:

$$\partial \mathbf{u}/\partial t + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla(p/\rho) + \nu \nabla^2 \mathbf{u}$$

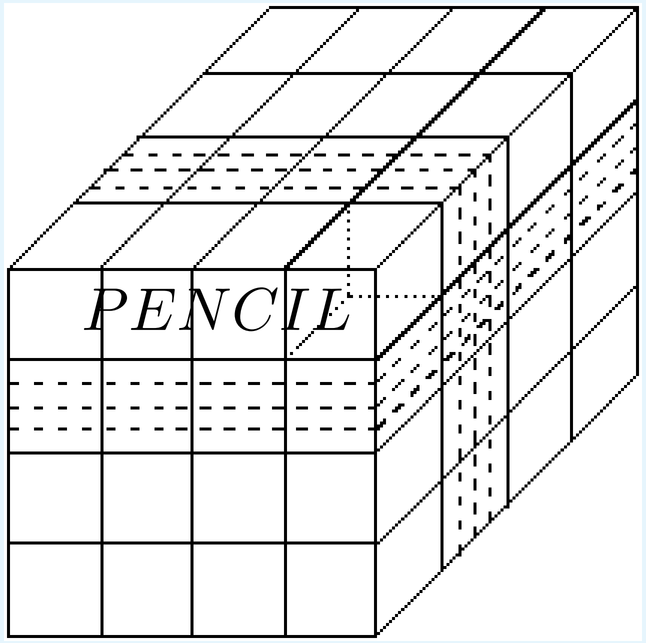
- In Fourier-space, $\mathbf{u}(\mathbf{x}) = \sum_{\mathbf{k}} \hat{\mathbf{u}}(\mathbf{k}) \exp(i\mathbf{k} \cdot \mathbf{x})$, with $\mathbf{k} \cdot \hat{\mathbf{u}} = 0$:

$$(\partial/\partial t + \nu k^2)\hat{\mathbf{u}} = -\{\widehat{\mathbf{u} \cdot \nabla \mathbf{u}}\}_{\perp \mathbf{k}}$$

- Explicit time stepping, viscous term by integrating factor
- $\widehat{\mathbf{u} \cdot \nabla \mathbf{u}}$ by convolution integral is impossible (Ops. $\propto N^6$)
- Fourier pseudo-spectral (beware aliasing errors):
  - forward and backward transforms, every time step
  - 3D FFT requirements (Ops. $\propto N^3 \ln_2 N$) drive the coding

2D Domain Decomposition

- Partition a cube along two directions, into “pencils” of data



- Up to N^2 cores for N^3 grid
- MPI: 2-D processor grid, $iproc(\text{rows}) \times jproc(\text{cols})$

3D FFT from physical space to wavenumber space:
(Starting with pencils in x)

- Transform in x
- Transpose to pencils in z
- Transform in z
- Transpose to pencils in y
- Transform in y

Transposes by message-passing, collective communication

PRAC Project Activities

- Approx 10 person-trips to NCSA (workshops and meetings)
- Use of BW Early Science System, May-June 2012
- Involvement in Track 1 acceptance testing (12288^3 benchmark)
- NEIS subaward from NCSA, 2012-2013
- Revised resource allocation granted by NSF, over 3 years
- Towards (first?) 8192^3 DNS of turbulence on periodic domain, with additional science requirements beyond velocity field

Factors Affecting Performance

Much more than the number of operations...

- Domain decomposition: the “processor grid geometry”
- Load balancing: are all CPU cores equally busy?
- Software libraries, compiler optimizations
- Computation: cache size and memory bandwidth, per core
- Communication: bandwidth and latency, per MPI task
- Memory copies due to non-contiguous messages
- I/O: filesystem speed and capacity; control of traffic jams
- Network topology of machine, environmental variables

Practice: job turnaround, scheduler policies, and CPU-hour economics

Benchmarking Protocols

- User-coded profiling (MPI_WTIME)
- Detailed breakdowns (major subroutines, or major classes of operations) are crucial for understanding
- Take max over all MPI tasks, and min over several steps
- To filter out effects of variability (from network contention), run different cases from same job (hence using same nodes)
- FFT kernel used to evaluate new programming strategies
  - key building block in PSDNS, also relevant to many other disciplines (open-source P3DFFT library, D. Pekurovsky)
  - specify a simple sinusoidal velocity field
  - transform to Fourier space, verify spectrum (single spike?)
  - transform back, max. error should be $O(10^{-6})$ or smaller

Subaward Objectives

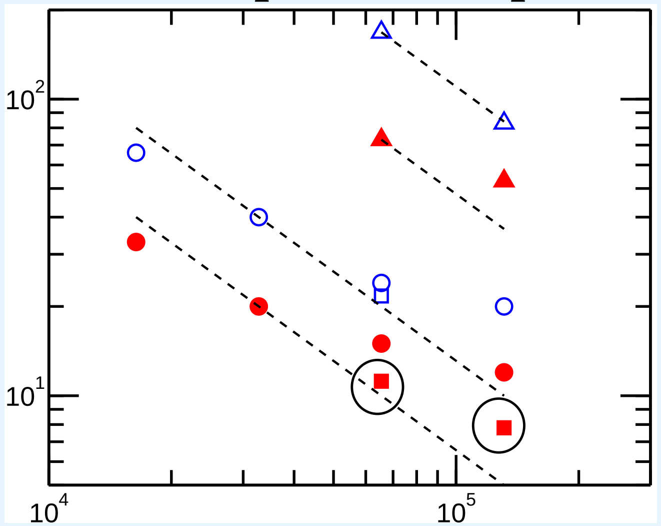
Top priority is improving communication performance:

- Hybrid MPI-OpenMP (multithreading) and overlap
  - production DNS code is fully hybridized, but only master threads make communication calls (while worker threads idle)
  - can we do better by overlapping communication by some threads with computation by other threads
- Co-Array Fortran and overlap
  - Alltoall by CAF (R.A. Fiedler) currently gives best performance, extended to other operations in DNS
  - can we do better yet by overlapping say, alltoall for one variable with computation for another variable

Production DNS Performance

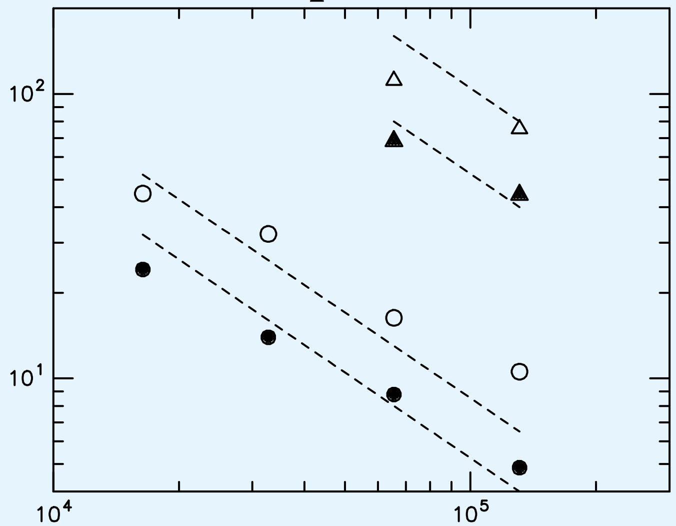
- 2+Petaflop Cray XK6 (Jaguarpf at ORNL) in Summer 2012 (similar to XE nodes on Blue Waters)
- 4096^3 (circles) and 8192^3 (triangles), 4th-order Runge-Kutta

CPU/step, MPI+OpenMP



# cores

CPU/step, MPI+CAF



# cores

- pure MPI, best processor grid, stride-1 arithmetic
- dealiasing: can skip some (high k) modes in Fourier space
- better scaling when scalars added (blue, more work/core)

Method 1: Hybrid MPI-OpenMP

- Parallel regions, shared or private variables, work-sharing constructs for computation
- 2 or 4 threads per MPI task (thread 0 is the master)
- Three possible levels of thread safety
  - FUNNELED: only master thread makes MPI calls (default, fully implemented in DNS code)
  - SERIALIZED: all threads can make MPI calls, but 1 thread at a time (use ORDERED construct)
  - MULTIPLE: all threads can make MPI calls, no restrictions
- Serialized and multiple: tested using FFT kernel only
  - Overlap: some threads compute while others communicate?
- Usually slower than pure MPI at small problem sizes, but more competitive for larger problem using more cores

Threads: serialized and multiple

Start with data in real-space, as pencils in x. Each thread will do: (a) FFT in x; (b) Pack; (c) ALLTOALL; (d) Unpack

- Serialized threads: a pipelined procedure

0	FFT in x	Pack	Alltoall	Unpack	......	......
1	FFT in x	Pack		Alltoall	Unpack	......
2	FFT in x	Pack			Alltoall	Unpack

  - Thread 1 waits until Thread 0 completes ALLTOALL
  - Unpack on thread 0 concurrent with comm on thread 1
  - Thread 2 in turn waits on thread 1; and 3 waits for 2

  Some penalty due to need for explicit synchronization

- Multiple threads: all threads doing ALLTOALL independently.
  - no explicit synchronizaion, but message traffic is heavy

FFT Kernel Performance

Time taken per forward-backward FFT for 5 variables:
(On BW, take best data from several repeat trials)

N^3	Cores	Tasks_Threads	CPU(secs, F/S/M)
2048^3	4096	16×128_2	3.65 / 3.03 / 2.99
2048^3	4096	8×128_4	5.20 / 4.27 / 3.74
4096^3	32768	16×1024_2	6.15 / 6.11 / 6.74
4096^3	32768	8×1024_4	6.75 / 6.58 / 7.40

- For comparison, pure MPI for same number of cores gives 3.20 secs for 2048^3 and 5.15 secs for 4096^3
- Relative merits of three modes (F/S/M) not clear, with significant variability between successive trials
- Because of NUMA considerations, > 4 threads usu. ineffective

Method 2: Co-Array Fortran

- Use Cray compiler (craycce): “ftn -h caf”
- Simple one-sided get operation for pairwise exchange, with random pair ordering
- In contrast to MPI_ALLTOALL, better to break messages into smaller chunks (512 bytes seems optimum).

```
complex(b8) :: recvbuck(buffersize)[0:*]
recvbuck(....)=src(...)
des(....)=recvbuck(...)[i_co]
sync memory
call mpi_barrier
```

- Declare major communication buffers as co-arrays: changes limited to a small number of routines but some copying is needed
- Needs huge memory pages (“*module load craype-hugepages8M*” and “*setenv XT_SYMMETRIC_HEAP_SIZE 200M*”)

Co-Array Fortran: Overlap?

Coarse-grain overlap for multi-variable FFT: exchange messages for one variable while computing another. Some challenges:

- For efficient overlap, buffer size needs to be large (hence try coarse-grain overlap instead of fine-grain)
- Make sure that CAF exchanges data in non-blocking manner (use *!dir$ pgas defer_sync*)
- Use multiple threads: one thread handles communication (including sync_memory) while others compute.
- How to synchronize the threads?
  - master thread exchanges data and wait for completion; worker threads to wait for signal from master. (use *!$OMP ATOMIC WRITE* to manage a shared variable for this purpose)

Q: Is it faster? A: Not yet, but preliminary timings comparable

Pseudo-code on overlap algorithm

```
flag_send_complete(:) = 0
!$OMP BARRIER
if (ithr==0) then ! Master thread
  sync memory
  call mpi_barrier
  do j=1,nv
    call CAF_alltoall (j)
    sync memory
    !$OMP ATOMIC write
    flag_send_complete (j)=1
  end do
else
  do j=1,nv
    do while (flag_send_complete(j) !=1)
      !$OMP FLUSH
      sleep
    end do
    !exchange completed for j-th variable
    !can proceed with computation
    call compute (j)
  end do
end if
```

Performance variability

- Substantial variability in timings on BW:
  - often large enough to obscure differences between different code versions or programming models
  - factor of 2 not uncommon (sometimes greater): precise estimation of resources required becomes difficult
- Major cause is understood to be network contention
  - shows in routines that perform communication
  - affects communication-intensive codes the most
- Possible solution, at systems level, is topology-aware scheduling:
  - encourage scheduler to assign nodes in close neighborhood
  - perhaps longer waiting time but leads to more efficient utilization of resources overall

Present timings on BW

For our largest jobs (8192^3, 4096+XE nodes), usually helps if:

- choose processor grid geometry such that $iproc \times num\_thr = 32$ (then some of the communication occurs on node)
- alltoall by CAF instead of MPI
- 1 thread, 16 MPI tasks/node (16 idle, more bandwidth/core)
- requesting a few percent more nodes than necessary
- favorable placement of nodes in 3D torus (most critical!)

Two jobs run with the same job script, 23 secs vs 54 secs/step:

taskid		itransform	realspace	transform		overall
0	0.249	8.694	0.390	12.24	1.32	22.89
0	0.249	22.35	0.394	29.70	1.68	54.11

I/O and Data Management

- Reading and writing checkpoints: (currently 1 file per MPI task)
  - alleviate traffic jam by a relay scheme:
    — at most 4096 MPI tasks doing I/O concurrently
  - organized into $\sim \sqrt{\mathrm{numtasks}}$ sub-directories
  - set directories to stripe 1 BEFORE writing data
  - on reading data, open files in read-only mode
  - file containing info on data organization also written
    (should be readable using different iproc or jproc)
  - conversion at input/output: overlap I/O with ....?
- I/O performance on shared Lustre filesystems can be highly variable, but seems very good on BW so far:
  - 40 secs to write 8192^3 single prec, velocity only (153 GB/s)
  - however reading takes longer (a concern for postprocessing)

I/O: Recent and Pending Changes

- Number of files is a weakness of current protocol
  - opening and closing a file puts load on metaserver
  - data transfer more efficient if fewer files

  But 1 file per simulation also cumbersome

- Working towards 1 file per communicator, w/ relay
  - basic POSIX format or Parallel HDF5

- Can we read/write less data?
  - since $\mathbf{k} \cdot \hat{\mathbf{u}} = 0$ some velocity Fourier coefficients can be recovered from others (implemented)
  - skip aliased modes: substantial reduction, but reading data with different iproc/jproc would be difficult (will need some effort)
  - checkpoint less frequently, or keep fewer datasets (maybe)

Data Archival and Processing

- Continuing for months or years:
  - retrieve the data from archival system (reliability)
  - re-analyze the data (as new questions, ideas come up...)
  - provide access to community (portability, formats)
- Globus Online used (only way?) to copy data between BW scratch disk and Nearline (HPSS) storage
  - 1000 Mbit/sec (125 MB/sec) sometimes possible
  - transfer proceeds in background
- However Globus Online does not provide sufficient functionality:
  - cannot move or list exact size of files in remote system
  - cannot verify file date, or change permissions

Concluding Remarks / Lessions Learned

- Facing up to the challenges of communication-intensive codes
  - bandwidth and interconnect topology
  - variability due to sharing of resources
  - OpenMP and (especially) Co-Array Fortran are helpful
- Pursuit of overlap between computation and communication in FFT kernels has so far not generated breakthroughs
  - yet there are still ideas to try
  - “dedicated box” testing on BW eagerly awaited
- With large resource allocation, and improvements to come:
  - Blue Waters will allow us to simulate and understand turbulence in unmatched precision and detail, providing great impetus for leading-edge HPC as well