HYCOM 2.0.01

Alan J. Wallcraft

Naval Research Laboratory

July 5, 2001

HYbrid Coordinate Ocean Model

- Based on MICOM
  - Miami Isopycnal Coordinate Ocean Model
  - Most widely used isopycnal ocean model
  - Starting from an isopycnal model greatly simplifies path to hybrid coordinates
- HYCOM Consortium for Data Assimilative Modeling
  - Goal: to develop and evaluate HYCOM
  - Multi-institutional effort under NOPP
    - UM/RSMAS; NRL; NOAA/AOML; U. Minnesota; LANL; SHOM; Planning Systems Inc.; Orbital Image Corp.; US Coast Guard
  - Provides “critical mass” to get HYCOM started
    - Contributions from the rest of the community are welcome
- Open Source ocean model
  - Include all additions from community, providing they don't break existing capabilities
  - Assimilation code will also be open source
  - http://panoramix.rsmas.miami.edu/hycom

HYCOM 2.0.01

- First public release of HYCOM 2.0
- More “main-stream” than MICOM
  - MKS units throughout
  - Default grid orientation is West to East then South to North
    - HYCOM 2.1 will allow arbitrary orientation (via input of lat-lon arrays)
  - User-tunable model parameters are read in at run time
- MICOM-like mode
  - Use HYCOM for pure isoycnal cases
  - HYCOM plot package also works with MICOM archive files
- KPP or Kraus-Turner mixed layer
- Energy-Loan (passive) ice model
- High frequency atmospheric forcing
- Scalability via OpenMP or MPI or both
  - Bit-for-bit multi-cpu reproducability

HYCOM 1.0 vs HYCOM 2.0 (I)

- Fortran 77 (MICOM-like) coding style in 1.0
  - Update to cleaner, Fortran 90 based, coding style
  - Retain much of the exiting Fortran 77 code
  - Dynamic memory allocation used by diagnostic programs
- New makefile setup in 2.0
  - Usage: `make hycom ARCH=sp3 TYPE=ompi`
    - ARCH is machine (E10K, alpha, o2k, sp3, t3e)
    - TYPE is method (one, omp, mpi, ompi)
    - Use TYPE=setup for diagnostic packages
- HYCOM 1.0 only for shared memory machines
  - Added MPI/SHMEM option
  - Either MPI or OpenMP or both
    - Single source code
    - Selectable at compile time
    - All “bit-for-bit” reproducable

HYCOM 1.0 vs HYCOM 2.0 (II)

- Periodic (global) regions not supported in 1.0
  - Adding halos for MPI automatically supports periodic boundaries
    - Near-global domains in HYCOM 2.0
  - Pan-Am grid requires a special halo exchange
    - Will be available in HYCOM 2.1
- Nested-domain open boundaries not in 1.0
  - Add 1-way nesting (as in MICOM)
    - In next release of HYCOM 2.0
  - Based on (new) archive files
  - Interpolate to target domain off-line
    - Source domain to target domain archive files
  - Model need only deal with one domain
    - Simplifies scalability
    - At the cost of more I/O and bigger files

HYCOM 1.0 vs HYCOM 2.0 (III)

- PAKK I/O not efficient or accurate
  - netCDF is eventual target
    - Not scalable or thread-safe
    - Would limit code portability if required by HYCOM
    - How to represent Pan-Am grid?
- HYCOM 2.0 reads/writes ".a and .b" files
  - ".a" is a raw IEEE REAL*4 array file (Fortran direct access)
  - ".b" is a plain-text header file (Fortran formatted)
- This I/O is simple and portable
- It can easily be parallelized
  - Have the N-th processor read/write every N-th 2-D array record
  - Require the record length to be a multiple of 16KB
- Convert to netCDF off-line

HIGH FREQUENCY ATMOSPHERIC FORCING

- MICOM use monthly climatological forcing
- Real forcing is 12-hrly to 3-hrly
  - Modeling actual calendar years
    - Essential for assimilation
    - Model day is days since 01/01/1901
    - Archive filename is: archv.YYYY_DDD_HH
  - Climatological forcing via 12-hrly anomalies
- HYCOM 2.0 (also HYCOM 1.0.10 and NLOM)
  - Just in time interpolation from native to model grid (outside the model)
    - Calculate the next model-run's fields while this run is in progress
  - Always hold 2 fields spanning the current time in memory
  - Interpolate these to current time

BIT-FOR-BIT MULTI-CPU REPRODUCABILITY

- Repeating a single processor run:
  - Produces identical results
- Repeating a multi-processor run:
  - Produces different results
    - Using either OpenMP or MPI
    - e.g. fastest global sum is non-reproducable
  - Unless programmer explicitly avoids non-reproducable operations
- Two levels of reproducability
  - On the same number of processors
    - Some scalable libraries provide this
  - On any number of processors
    - Only “safe” option for code maintenance
    - Always requires carefull programming
    - Can be slower
    - Should be required for operational ocean prediction models
    - Is required by HYCOM

ARE TWO HYCOM RUNS IDENTICAL?

- The only way to comfirm bit-for-bit identity is to compare binary fields
- Could compare binary archive and/or restart files
  - But these don't tell you where any differences came from
- P-MICOM used “named pipes” to compare arrays between MASTER and SLAVE model runs while they were in progress
  - A named pipe is a special Unix file providing a FIFO capability
  - Can read and write to it just like a normal file
- SLAVE writes an array to the pipe, MASTER reads the array and compares it to its own version
  - Usually MASTER runs on one processor and SLAVE on multiple processors
  - Only limitation is that MASTER and SLAVE must be running under the same Unix image
    - May be difficult to arrange for MPI on a cluster

ARE TWO HYCOM RUNS IDENTICAL? (continued)

- HYCOM includes a named pipe based comparitor
  - Similar to P-MICOM, but easier to use
- Include calls to compare or compareall in source code
  - These will trigger a comparison at run time if the named pipe exists
- Standard scripts available to demonstrate how this works
- Used to debug OpenMP logic
  - Found differences that occured only about every 100-th time step
- It is possible for the additional synchronization from the exchange to hide subtle bugs
  - But this isn't likely
- Each new release of HYCOM is tested for multi-cpu reproducability on several machines using named pipes
  - Run your own tests using the provided scripts

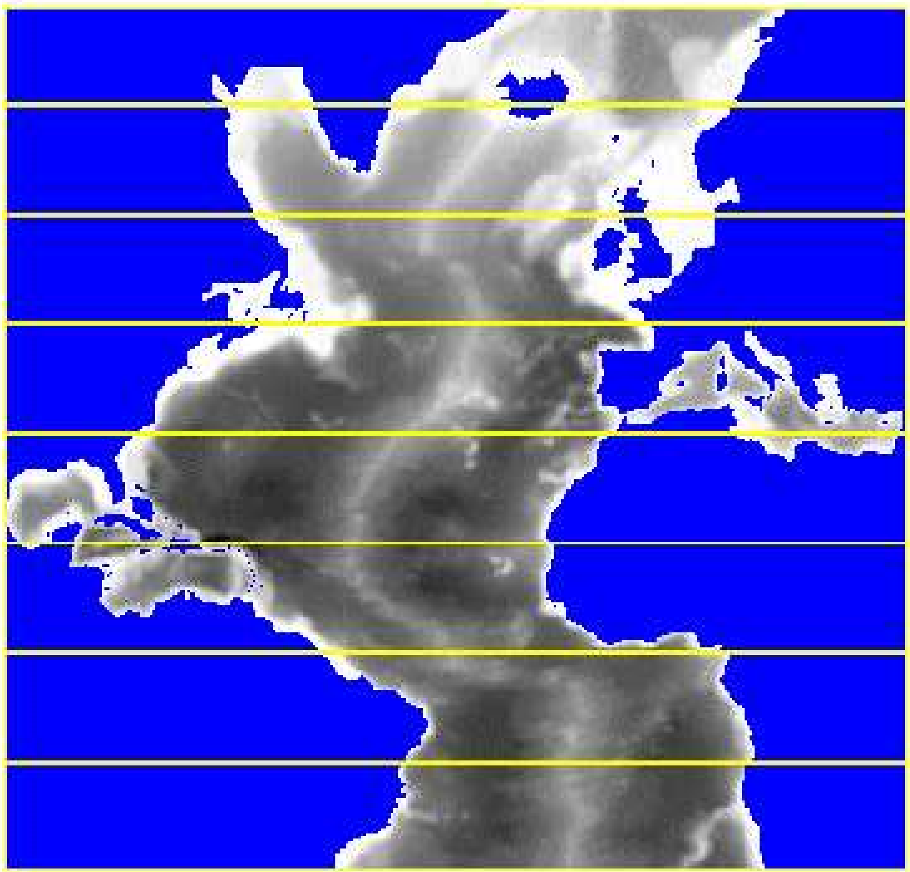
HYCOM OPENMP

- OpenMP Fortran is a standard set of directives for parallelizing loop nests
- Multiple threads work simultaneously on the same loop
  - Each loop iteration is run by exactly one thread
  - Set of all iterations distributed between threads
  - Typically can't go to the next loop until all threads have completed all their iterations of this loop
- OpenMP HYCOM parallelizes the j-loop (latitude)
  - Reordered loops to maximize work per j-loop
  - Bit-for-bit multi-cpu reproducability of global sums implemented by first forming zonal sums in parallel and then summing these in a serial loop
- Each latitude line has a different amount of ocean (i.e. a different amount of work)
  - So balancing the per-thread load is important
  - Otherwise some threads spin, waiting for others to complete the loop

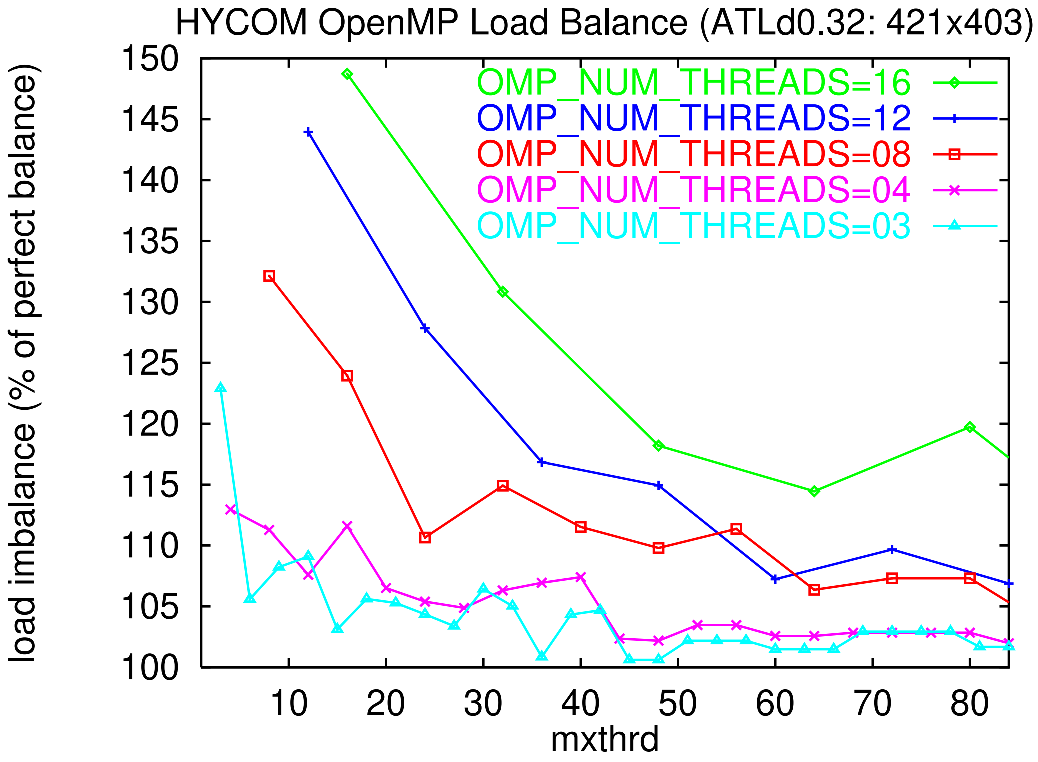
HYCOM OPENMP (continued)

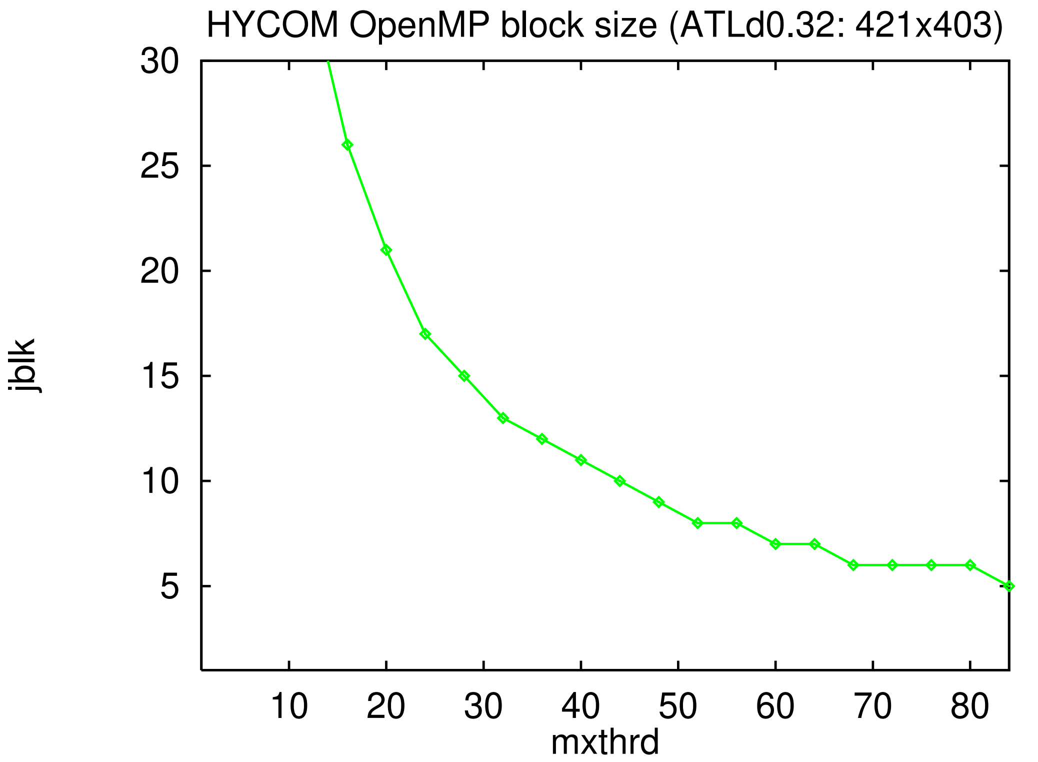
- OpenMP provides several SCHEDULE (iteration distribution) options
  - Default is vendor-specific
    - Portable programs must specify a SCHEDULE
- However, none are optimal for HYCOM
  - Static schedules don't load balance
  - Dynamic schedules have poor cache performance
- HYCOM uses SCHEDULE(STATIC,jblk)
  - The chuncksize, jblk, is chosen at compile time to give mxthrd chunks
  - The number of chunks, mxthrd, is typically a multiple of the actual number of threads used
- Setting mxthrd $>>$ OMP_NUM_THREADS creates an interleaved distrubution
- A larger mxthrd gives better load balance, and a smaller jblk (good for cache performance)
  - But a very small jblk increases cache sharing
- Set mxthrd on a case by case basis

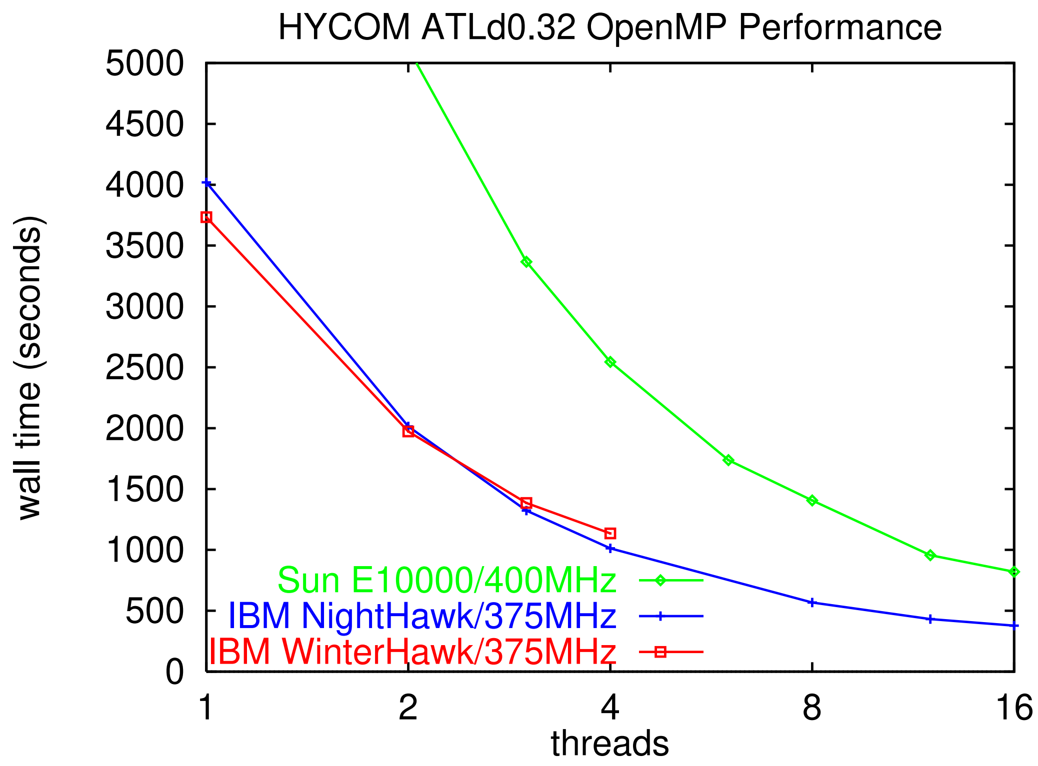
OpenMP SCHEDULE(STATIC,51)

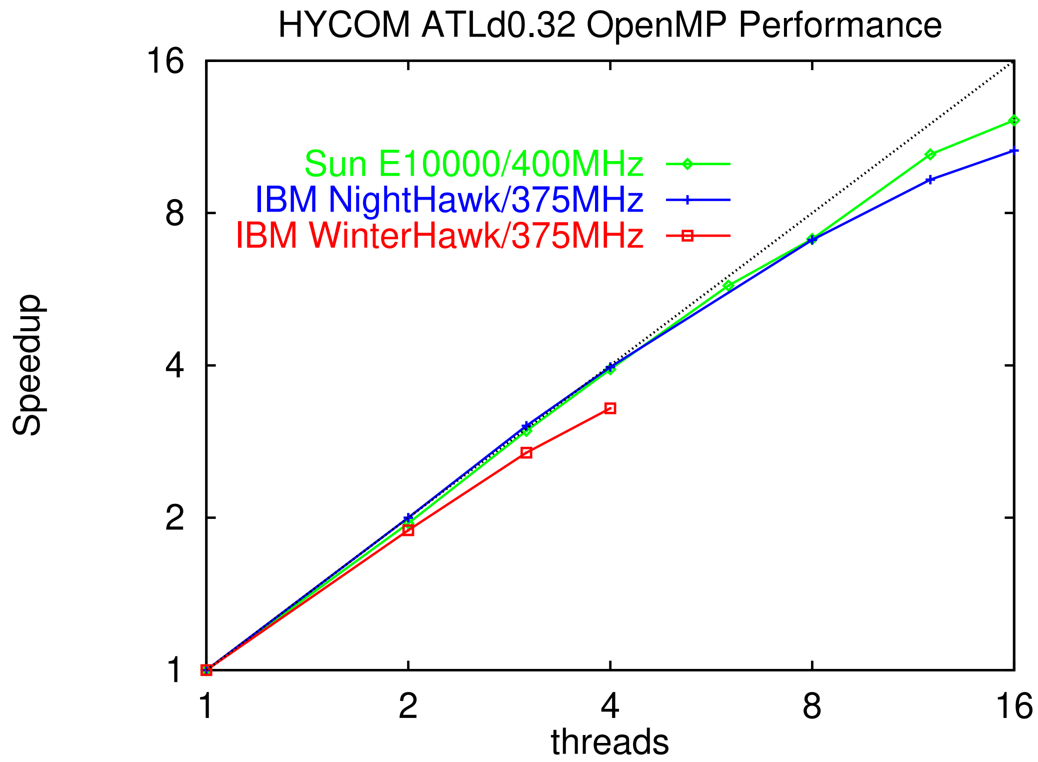


69%	99%
106%	102%
94%	89%
132%	111%
129%	
97%	
84%	
90%	


HYCOM OpenMP Load Balance (ATLd0.32: 421x403)
load imbalance (% of perfect balance)
mxthrd
OMP_NUM_THREADS=16
OMP_NUM_THREADS=12
OMP_NUM_THREADS=08
OMP_NUM_THREADS=04
OMP_NUM_THREADS=03


HYCOM OpenMP block size (ATLd0.32: 421x403)
jblk
mxthrd


HYCOM ATLd0.32 OpenMP Performance
wall time (seconds)
threads
Sun E10000/400MHz
IBM NightHawk/375MHz
IBM WinterHawk/375MHz
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1
2
4
8
16


HYCOM ATLd0.32 OpenMP Performance
Speedup
threads
Sun E10000/400MHz
IBM NightHawk/375MHz
IBM WinterHawk/375MHz
16
8
4
2
1
1
2
4
8
16

DOMAIN DECOMPOSITION

- Given a time dependent problem over a spatial domain where most operations per time step are local
  - Ocean and Atmosphere models
  - Computational Fluid Dynamics
  - Oil Reservoir models
- Split the domain into contiguous sub-domains
  - Size each sub-domain for equal work and minimal connectivity to other sub-domains
  - Structured or unstructured grids
- Add a “halo” or “ghost cells” around each sub-domain such that:
  - If the halo is up to date;
  - Sub-domain operations are independent
    - Only using sub-domain and halo values
- Program only has memory for one sub-domain plus its halo
- Domain is distributed across the processors
  - Communicate via MPI or SHMEM

EQUAL-SIZED RECTANGULAR TILES

- Simplest scheme is equal-sized rectangular tiles
  - Identical to data parallel block layout
  - Each tile has four nieghbors
    - Eight nieghbors including halo corners
  - Each row might exclusivly map to SMP nodes
- Overall speed controled by slowest tile
  - Probably have an “all ocean” tile
    - no advantage to avoiding land
- So, discard tiles that are entirely over land
  - Relatively simple to implement
  - Does not discard all land
  - Better for large tile counts
  - Ineffective on very small tile counts
  - Destroys any row to SMP node mapping
  - P-MICOM

STRUCTURED VS UNSTRUCTURED DOMAIN DECOMPOSITION

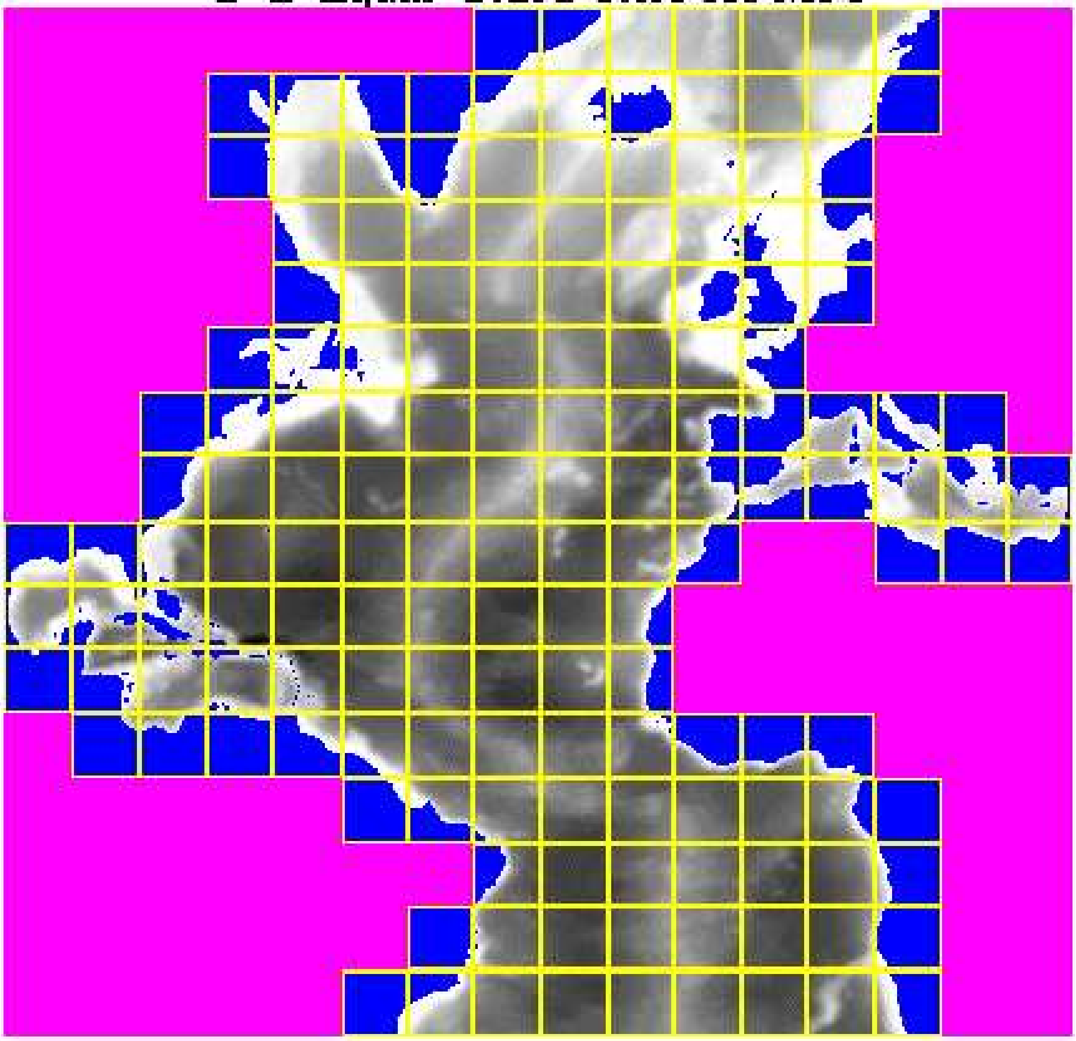
- Structured grid domain decomposition
  - Equal size rectangular tiles
  - Connected to only four other tiles
  - Can't avoid all land
- Unstructured grid domain decomposition
  - Irregularly shaped sub-domains
  - Connected to many other sub-domains
  - Near perfect load balance
- Relax constraints on structured grid decomposition to approach unstructured grid land efficiency
- Explore “separable” decompositions
  - Decompose each axis separately
  - Still get rectangular tiling
  - All tiles in same row are equal height
  - Two East-West nieghbors
  - Many North-South nieghbors

EQUAL-OCEAN RECTANGULAR TILES

- Each rectangle contains the same amount of ocean
  - Near perfect load balance
    - Work per ocean point is approximately constant
    - Must skip all land calculations
  - More expensive halo exchange
    - Many North-South nieghbors
    - Variable tile size
- Some tiles require more memory than others
  - If they contain a lot of land
  - Can “shrinkwrap” tiles to reduce memory
  - Memory requirement may set minimum tile count
- Aspect ratio of rectangle can be large
  - Replace one N tile row with two N/2 tile rows
  - Can be coded as discarding N/2 tiles per row
- Equal-ocean code can easily handle equal-area tiles
  - Choose whichever is most appropriate
  - HYCOM 2.0

MICOM 2–D DOMAIN DECOMPOSITION

2–D Equal–Sized Tiles for MPI



Each Tile has 4 neighbors

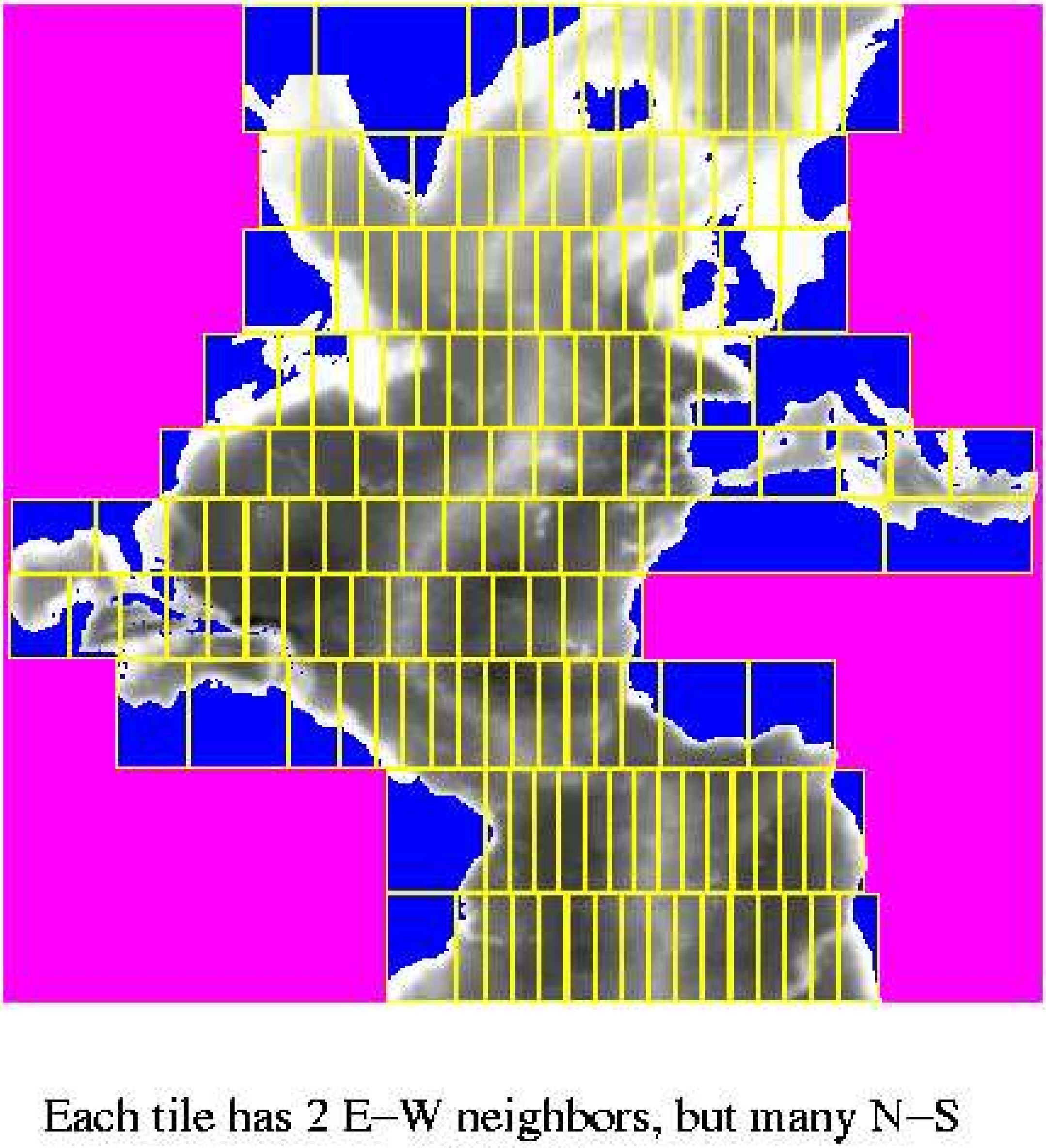
Discard "All–Land" Tiles

16x16 = 256 Tiles but only 161 active

Still have significant load imbalance

HYCOM 2–D DOMAIN DECOMPOSITION

2–D "Equal Ocean" Tiles for MPI



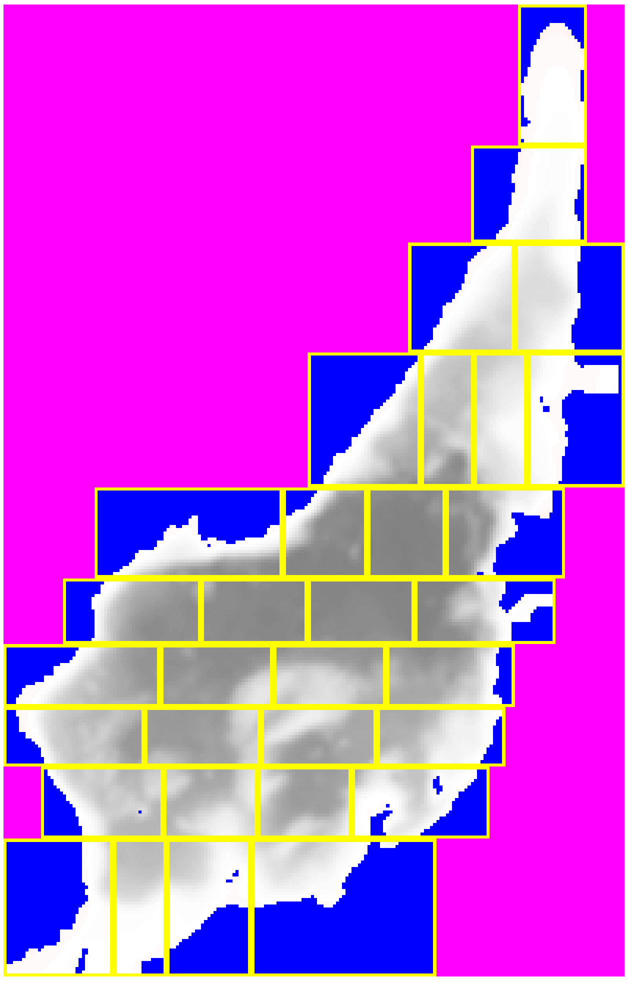
Each tile has 2 E–W neighbors, but many N–S neighbors

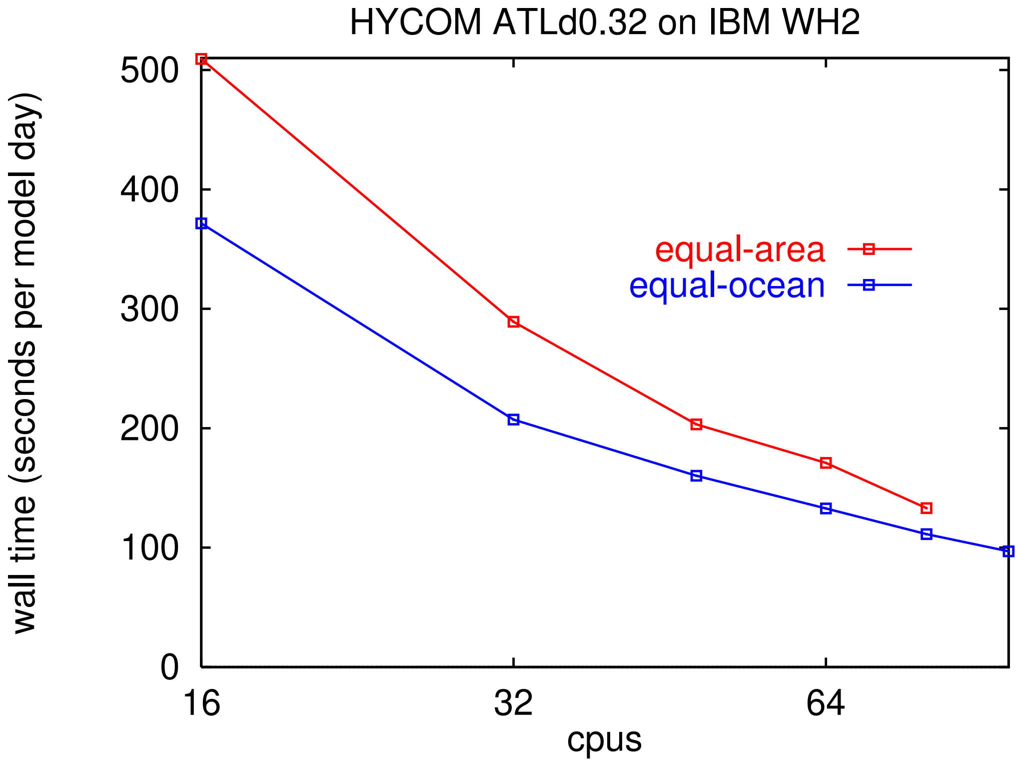
Compared to 16 x 16 "equal – size" decomposition:

30% less work on "worst" tile

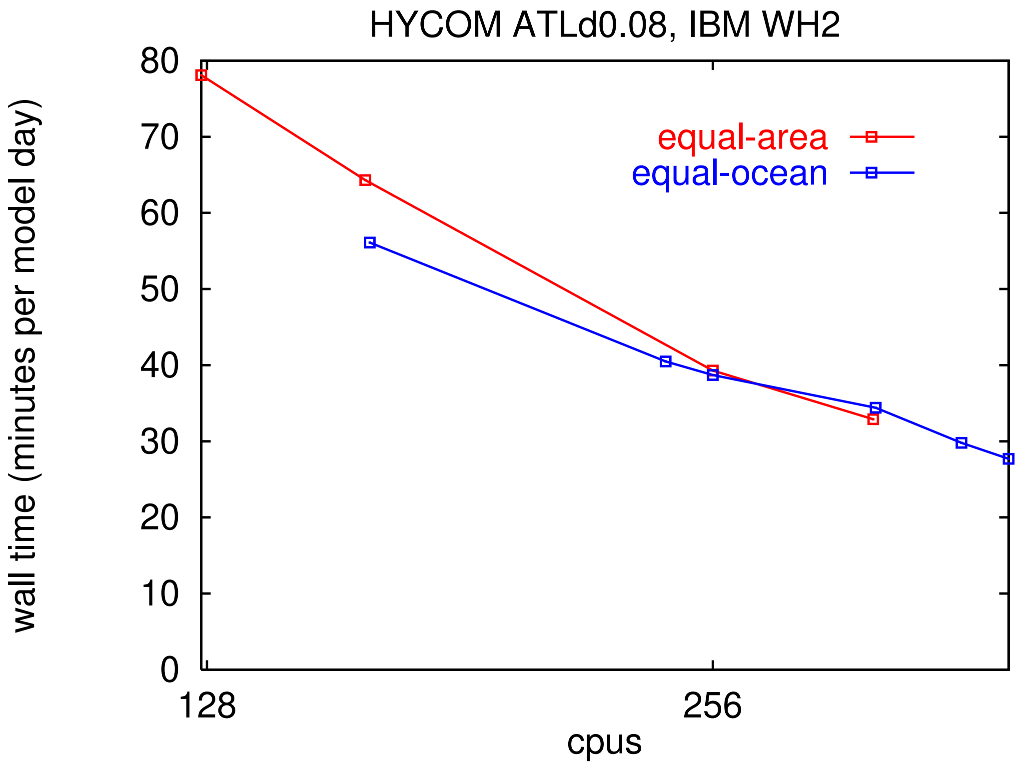
larger memory requirement

HYCOM JES TILING
(4x8 tiles, top row becomes 3 rows)





HYCOM ATLd0.32 on IBM WH2
wall time (seconds per model day)
500
400
300
200
100
0
16
32
64
cpus
equal-area
equal-ocean



HYCOM ATLd0.08, IBM WH2
wall time (minutes per model day)
80
70
60
50
40
30
20
10
0
128
256
cpus
equal-area
equal-ocean