Earth System Modeling and Computer Performance

Patrick H. Worley

Computational Earth Sciences Group

Co-lead of ACME Performance Group

June 9, 2015

CCSI Earth System Modeling Workshop

Oak Ridge National Laboratory

Oak Ridge, TN

ORNL is managed by UT-Battelle
for the US Department of Energy

Climate Change Science Institute
Oak Ridge National Laboratory

Acknowledgements

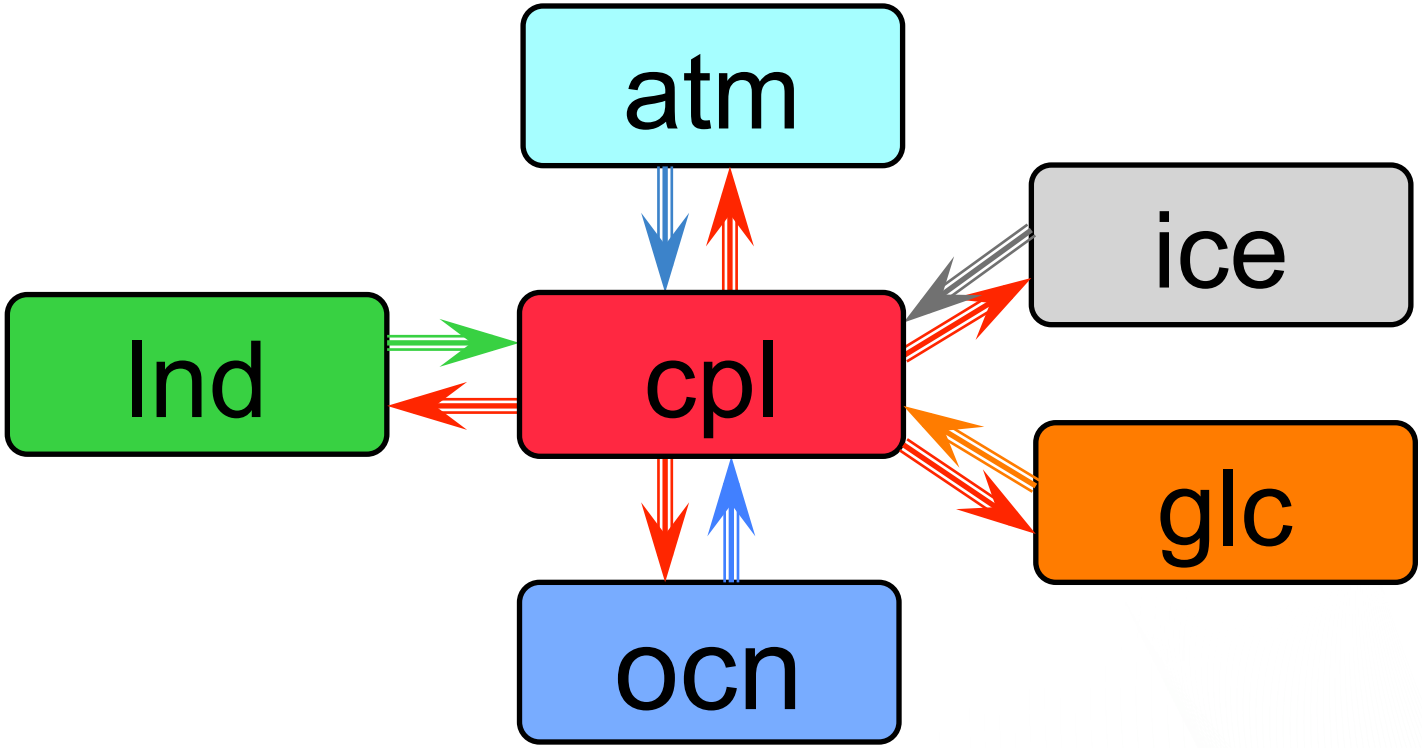
- This research was sponsored by the Office of Advanced Scientific Computing Research and by the Office of Biological and Environmental Research, U.S. Department of Energy, as part of the projects "Accelerated Climate Modeling for Energy (ACME)", "Applying Computationally Efficient Schemes for BioGeochemical Cycles (ACES4BGC)", and "Institute for Sustained Performance, Energy, and Resilience (SUPER)". The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy.

- An award of computer time was provided by the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.This research also used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under contract DE-AC02-06CH11357. Finally, this research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Background

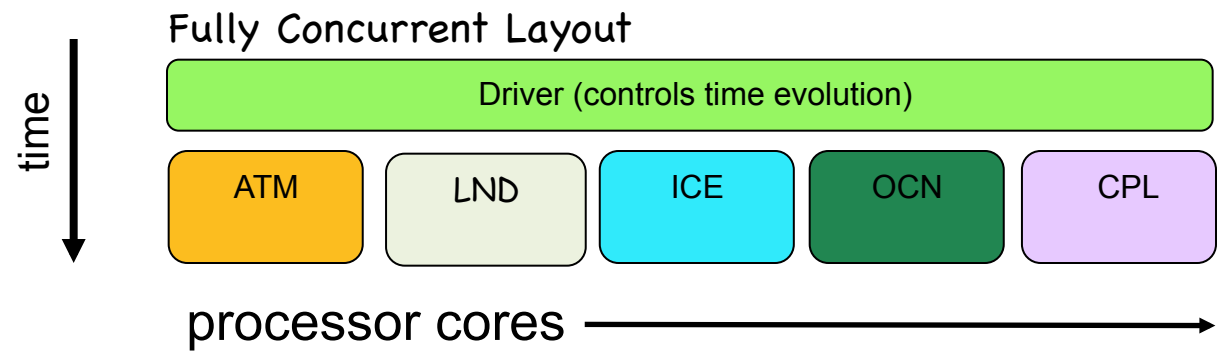
- Focusing on the Accelerated Climate Model for Energy (ACME)
  - DOE model that branched off of the DOE/NSF/xxx Community Earth System Model (CESM) in August 2014. ACME and CESM are coordinating development, so they will continue to be similar in many ways going forward.
- ACME (and CESM)
  - Consists of a set of 5 (or more) geophysical component models on potentially different grids that exchange boundary data with each other via communication with a coupler (hub and spoke architecture)
  - Large code base: >1M lines
    - Fortran 90 (mostly), but new development is also incorporating libraries written in other languages (e.g., Trilinos in C++ and PETSc in C).
    - Developed over nearly 30 years, but much code is relatively new.
    - 200-300K lines are "critically important" --> no dominant kernels; need good compilers and a performance portable programming style.
  - Parallel implementation utilizes both MPI and OpenMP, and OpenACC or CUDA Fortran for GPU support.

ACME / CESM Hub and Spoke

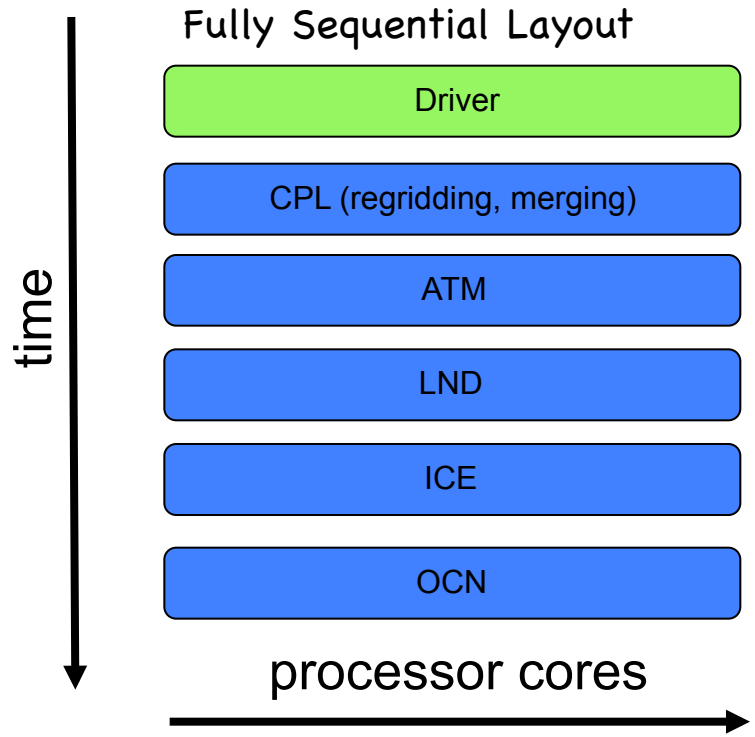


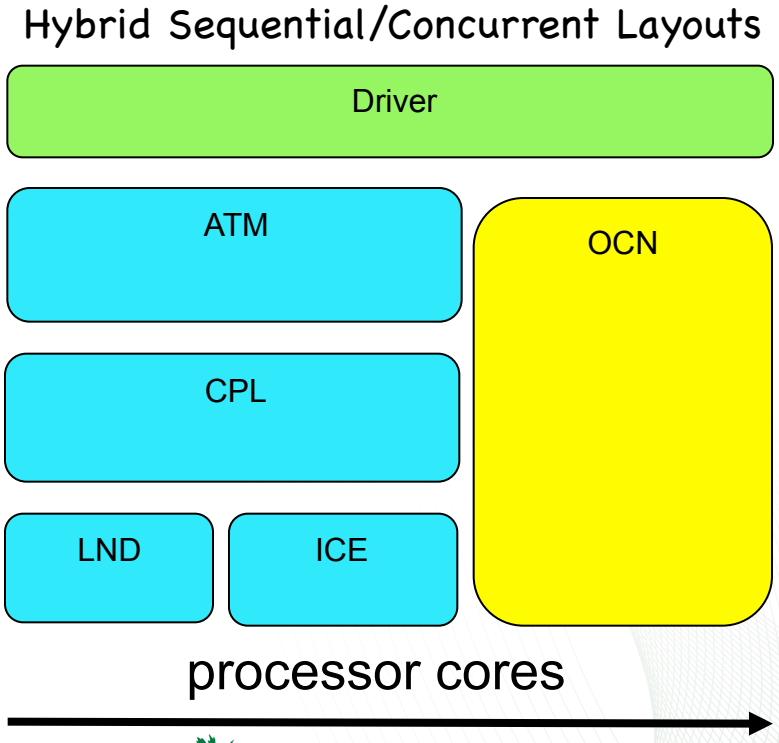
- Logical view of coupling architecture: atm (atmosphere), ice (sea ice), glc (land ice), ocn (ocean), lnd (land). (Missing rof (river transport) and wav (wave) components.)
- Cartoon of parallel implementation in concurrent mode.

ACME / CESM Architecture



Note that this is an old figure: missing GLC, ROF, and WAV.





Performance Metrics: Definitions

- Simulated Years Per Day (SYPD)

  The number of simulation years that a model calculates when running for a single "wall clock" day

  – for some science case

  – on a particular computer system

  – with some amount of computer system resources

  This is a measure of (or bound on) how much "work" you can get done in a day.

- (Processor) Core Hours Per Simulated Year (CHPSY)

  The number of processor cores times the number of hours when calculating a single simulation year. This is a function of the same parameters as SYPD.

  This determines how quickly you will use up your computer allocation for your simulation runs. A good estimate of this is important when requesting the allocation size in a proposal.

  In reality the cost is in "node hours" when running systems where applications do not share compute nodes, but allocations are often in core hours anyway.

Performance Metrics: Large Example

1. Coupled model run using ACME:
   a. Active atmosphere, ocean, land, and sea ice on a 1/4 degree horizontal resolution atmosphere/land grid and a 1/10 degree horizontal resolution ocean/sea ice grid; 30 vertical levels in ATM and 42 in OCN.
   b. Simulated 21 months.
   c. Job ran for 23 hours and 40 minutes.
   d. Used 67,916 processor cores of Titan.

   So, the throughput rate for this job was

   – 1.8 SYPD .

   Using 67,916 cores requires 4245 nodes on Titan – each node has 16 cores, leaving the last node with 4 idle cores. The Titan charging algorithm is 30 cores per node (representing the 16 processor cores + one GPU accelerator), so cost was

   – 3,014,375 core hours for this one job,
   – 1,722,500 CHPSY for this case and node count.

Performance Metrics: Small Example

2. Coupled model run using ACES4BGC project branch of CESM:
   a. Active atmosphere + extra chemistry, ocean, land, and sea ice on 1 degree horizontal resolution component grids; 30 vertical levels in ATM and 42 in OCN.
   b. Simulated 2 years.
   c. Job ran for 5 hours.
   d. Used 3,088 processor cores of Titan.

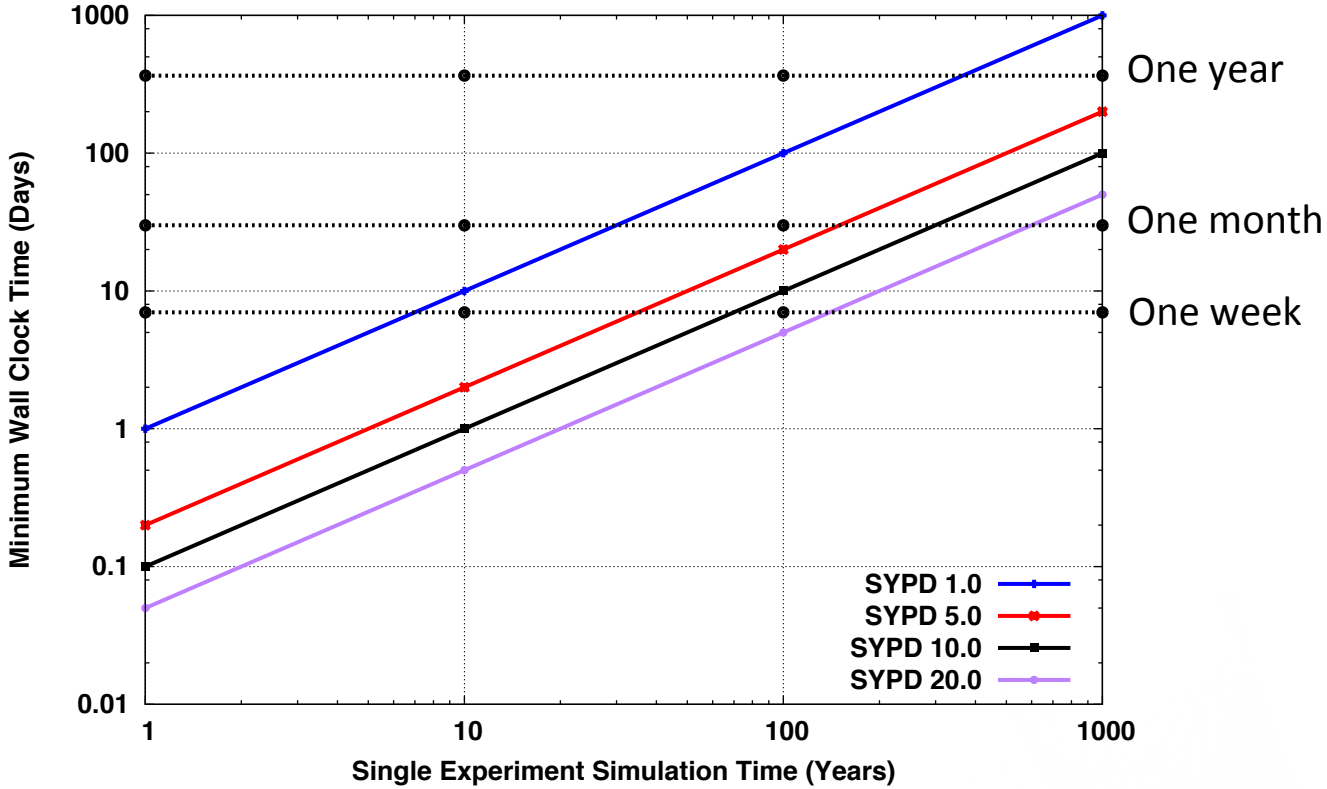
   So, the throughput rate for this job was

   – 9.6 SYPD .

   Using 3,088 cores requires 193 nodes on Titan. With the Titan charging algorithm of 30 cores per node the cost was

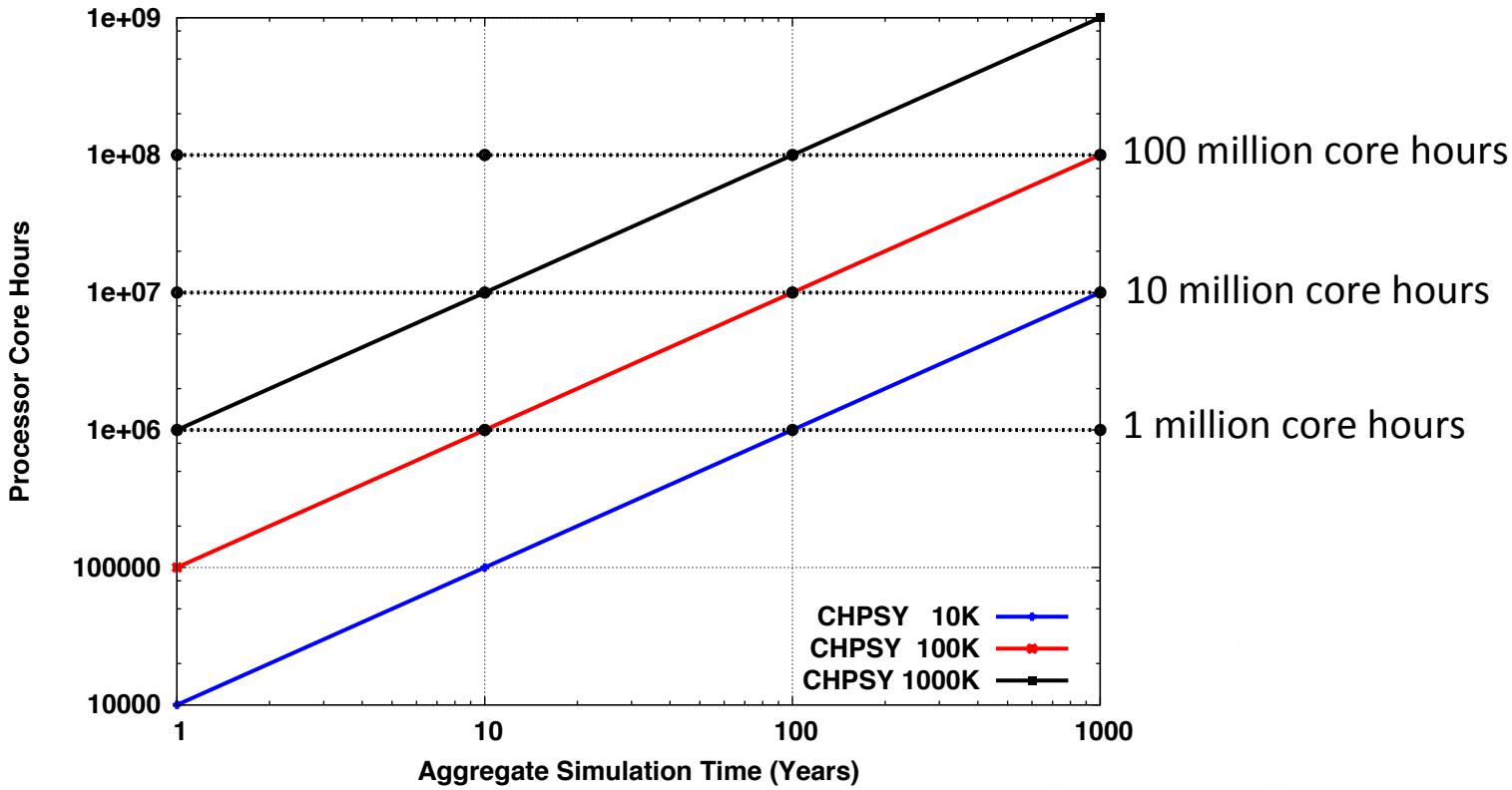
   – 28,950 core hours for this one job,
   – 14,475 CHPSY for this case and node count.

Why Should You Care: Runtime



- A simulation from present day to 2100 will take 8.5 “real” days at a rate of 10.0 SYPD, and almost 3 months at a rate of 1.0 SYPD.
- This is the lower bound on the throughput of a single simulation. An ensemble with, e.g., an aggregate of 1000 simulation years is constrained only by the duration of a single member (job) if the members can run concurrently.

Why Should You Care: Allocation

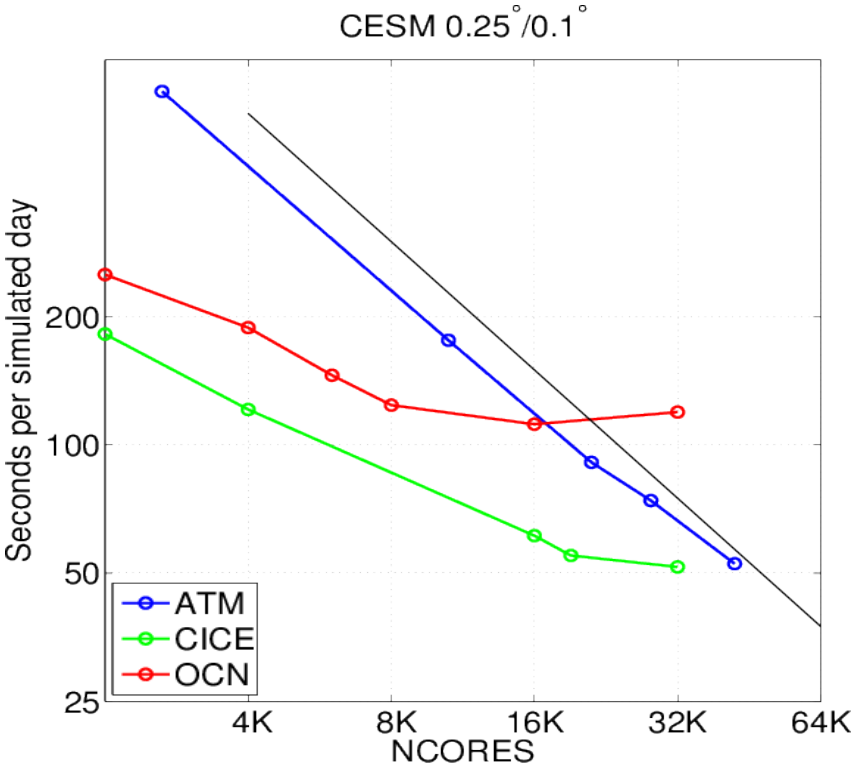


- Largest allocation on Titan in 2015 is 280 million core hours.
- Example 1 is almost 2000K CHPSY on Titan, and would require 100 million core hours to simulate 50 years (aggregate), say a 5 member ensemble of 10 SY each.
- Example 2 is 15K CHPSY on Titan, so can simulate over 6500 simulation years with 100 million core hours (but taking almost 2 years if a single simulation).

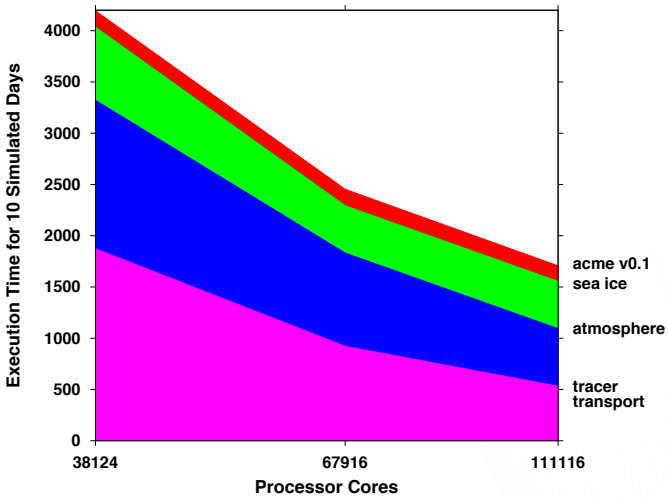
Why Should You Care: Implications

- The execution time of individual jobs influence (or constrain) the amount of science that gets done (and the number of papers written) during the duration of a project (and during a researcher's career).
  - If the science is "better" in an expensive simulation job than in a different, cheaper, job, perhaps because of a higher resolution grid or different enabled physics or more frequent output, then this analysis is simplistic.
- The more expensive, in core hours, that individual jobs are, the fewer jobs can be run for a given allocation. This may decrease the science output for a project (all else being equal), and/or decrease the available resources for other projects.
- So,
  - making a target science job run more quickly may accelerate the science, if the allocation is sufficiently large to support all required experiments;
  - making a target science job run using fewer core hours enables more runs for a fixed allocation, which may accelerate the science if the execution time is sufficient for all jobs to complete "quickly enough";
  - making a target science job run more quickly on similar, or fewer, resources may accelerate the science (without qualifiers).

Complications: Scaling

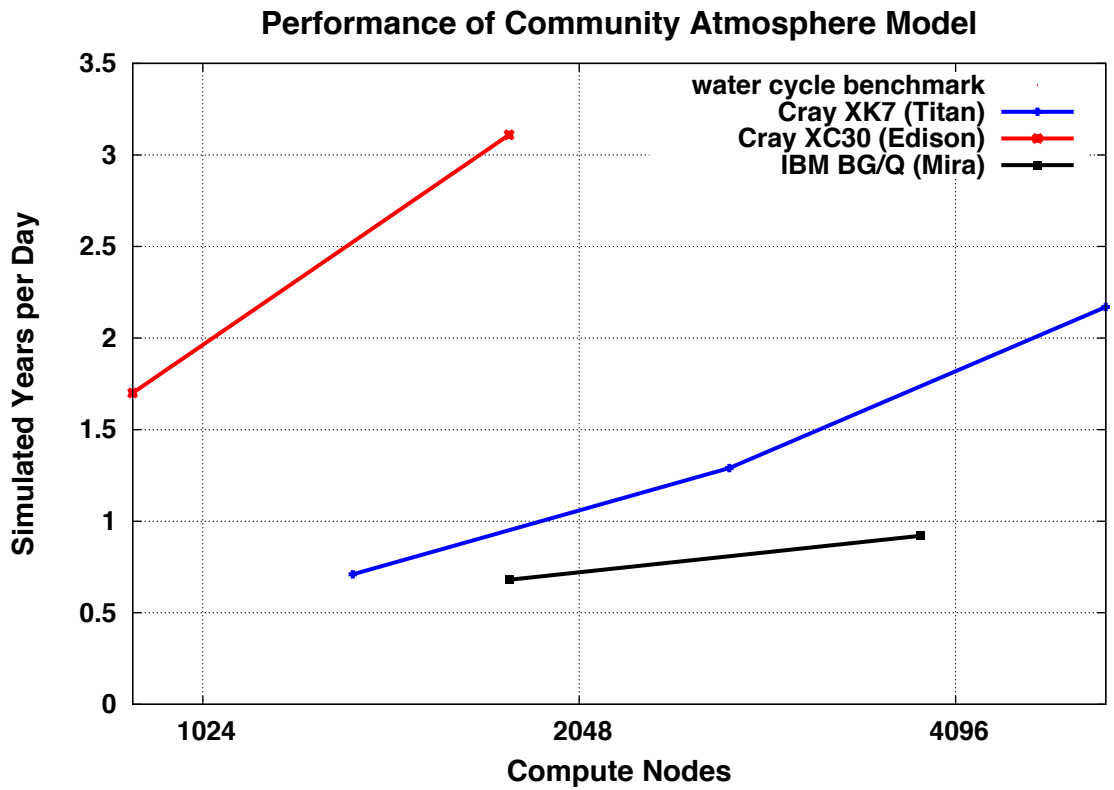


- Left: Strong scaling of ACME components running Example 1 on Titan. (Courtesy of M. Taylor.)
- Below: Example strong scaling of ACME (with performance limiting components) with more vertical levels (30 -> 64) and tracers (25 -> 50) than Example 1.



- Increasing the number of cores will not typically decrease the runtime by the same factor. These are fixed size problems (strong scaling), and communication overhead and load imbalance will decrease parallel efficiency, and at some point there will also not be enough work to keep all of the cores busy. The details also vary from component to component.

Complications: Systems



- Left: Strong scaling of ACME atmosphere when running Example 1 with additional vertical levels (30 -> 64) and tracers (25 -> 50).

- Different computer systems have different capabilities (when running ACME and CESM). It is often difficult to compare between systems because processor, node, network, and I/O architectures can be very different. Other factors are also important – size of allocation, maximum system size, number of users on the system, batch queue priorities and caps, …

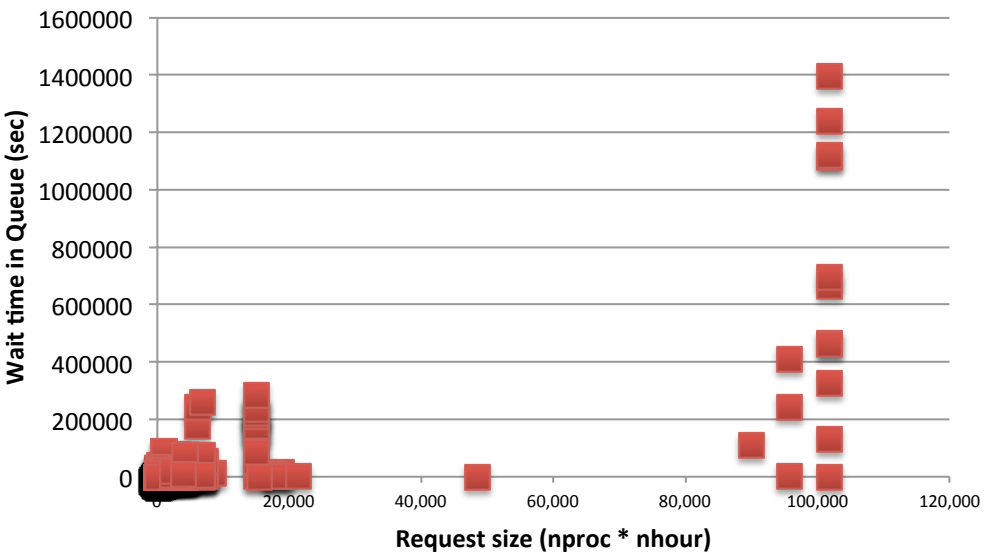
Complications: Queue Time

- Job execution time can be only a small component of job throughput. For example, on Titan:
  - Large node count jobs receive priority, but also compete with other large jobs. There is also a cap on execution time of 24 hours for a single job (without making special arrangements).
  - Full system jobs may run only once a week.
  - Small node count jobs have lower priority, and also have smaller caps on maximum runtime, so a 24-hour execution might require submitting four 6-hour jobs, with some amount of queue time for each job when none of the jobs are running.

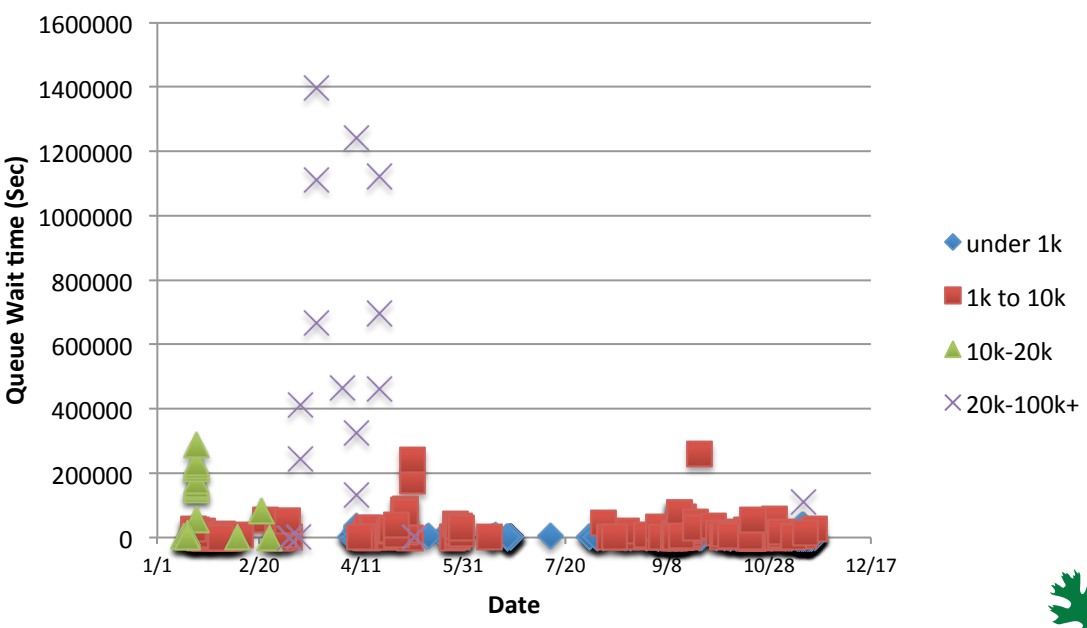
  So execution time and core hours are parameters in determining science throughput, but not in as simple a way as described previously. (The simpler analyses are still lower bounds.) In particular, the system workload (jobs from all users) also constrains how quickly your jobs will be scheduled.

  However, the less execution time that a simulation requires, the fewer jobs that it needs to be divided up into, and the less time is spent waiting in the queue (in general).

Complications: Queue Time



- Time spent in queue as a function of requested core hours during 2014 for 485 CESM jobs. (Figure courtesy of B. Mayer.)



- Same data as above, but also indicating when jobs were submitted. Contention for resources is higher during certain times of the year, e.g. just prior to Supercomputing paper and Gordon Bell submission deadlines. (Figure courtesy of B. Mayer.)

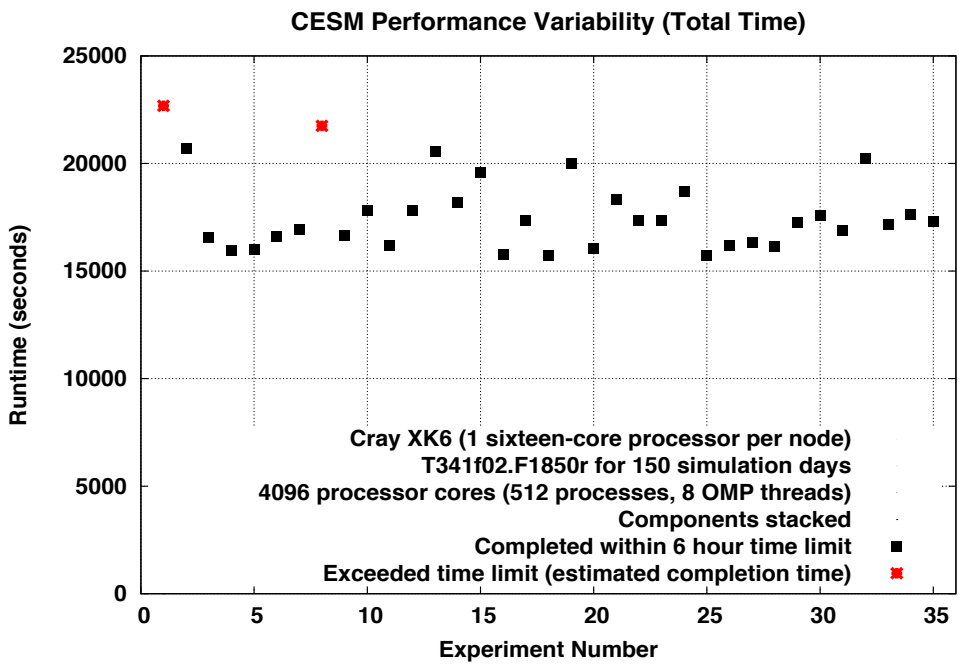
Complications: Others

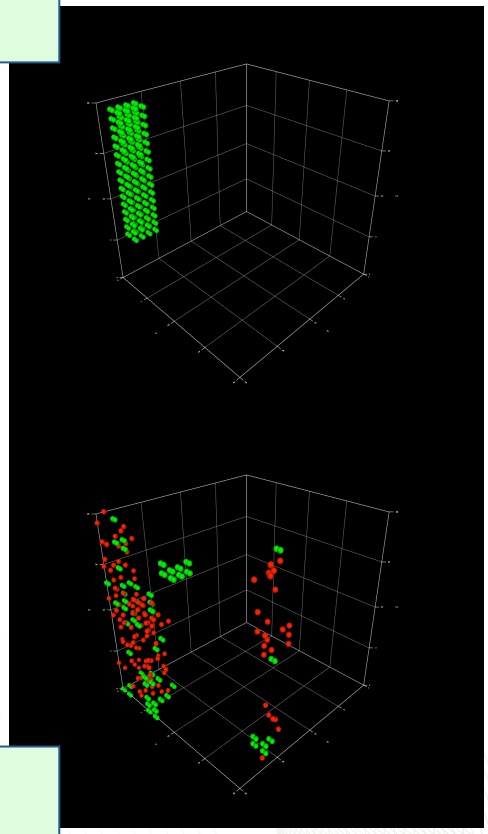
- Performance Variability
  - Execution time of a job can vary depending on placement of job in relation to other running jobs (contention for interconnect) and contention with other users for parallel file system. Impact is exacerbated if job exceeds requested wall clock time, e.g. 24 hours, and is killed, thus requiring repeat of simulation since last checkpoint write. Typically requires manual postprocessing to resubmit, which introduces further delay.
- Failures
  - They happen - failure during program load, HW/SW failure in middle of simulation, failure when reading or writing to the file system, … Same impact as above, requiring manual postprocessing, resubmission, and possible recomputation from end of last successful checkpoint.
- Initialization Time
  - Each job has an associated start up cost independent of the simulation time, and shorter jobs will spend a larger percentage of time on this. For example, Example 1 can take up to 20 minutes to initialize.

Complications: Performance Variability

Example: Variability in performance within a suite of climate model experiments on the Cray XK6 (earlier version of Titan) at ORNL in 2012.

Allocated nodes for fastest run.





Allocated nodes for slowest run. Red nodes share a network switch with a different job.

Complications: Example

- Example 1 is one job in a suite of jobs representing a single long simulation on Titan. For members of this suite starting on or after April 20, 2015 …
  - Each job uses 67,916 cores (4245 nodes), requests 24 hours, and attempts to simulate 21 months.
  - 16 jobs have run through June 7, 2015, of which 7 completed successful and a further 7 completed at least one checkpoint before terminating.
  - 19.8 years have been simulated successfully (at the cost of simulating 23.1 years due to the failures, so 17% extra).
  - Effective SYPD is 0.41 (= 19.8 SY / 48 days). In comparison, the average per job throughput rate is around 2.0 SYPD.

Mitigation Strategies

- What if the execution time and/or the core-hours is too large, and the target science can not be achieved in the allotted time with the allotted allocation?

  (first things to try; anyone can do, in theory)

  - Optimize layout (processes/threads per component, and location of components)
  - Optimize runtime performance options

  (more extreme, requiring specific expertise in some cases)

  - Revisit problem formulation
    - Can you get by with less resolution or less expensive physics and still address the target science?
  - Novel or alternative algorithms, e.g.
    - Ensemble- based alternatives to long simulations
  - Further optimization of performance critical code (see next slide)
  - Move to a different platform
  - …

ACME Performance Group Activities

1. Preparing ACME (project and code) for next generation architectures
2. Evaluating and optimizing ACME v1.0 for target systems
3. Advising on and optimizing performance for ACME development runs

Activity #2 includes:

- identifying and vectorizing computational hotspots
- evaluating, enhancing, and optimizing (OpenMP) threading
- porting to and optimizing performance-critical components on GPUs
- evaluating and optimizing MPI communication and load balance
- developing, evaluating, and optimizing new parallel I/O infratructure

for all ACME components, as well as for the full coupled model, on Edison, Mira, and Titan.

What Can You Do?

1. Code performance optimization is complicated, if only because you need to avoid breaking the code. Even changing compiler flags, especially those related to code optimization, can break the code. So adequate testing is critical, and code performance optimization is best done in coordination with the model development community.

2. Component layout optimization is something every user can do, with some practice. The defaults are not guaranteed to be good performers, especially for nonstandard cases and/or on the high(er) performance computing systems. (See next slides.)
   - Once you choose a layout, it is often best to stick with this for an experimental "campaign". Changing the layout is likely not bit-for-bit, due to changes in arithmetic ordering, and these perturbations will make the simulation diverge from that using a different layout. The climate statistics (should) still be correct, but the particulars will not be identical.
   - There are new tools to automate the selection of a good layout, and the manual method described next may not be necessary going forward.

3. Optimizing over runtime performance options is also something every user can do, but likewise requires some training.

Optimizing Component Layout

- The following slides are specific to ACME/CESM, and are a very simple overview. To be actionable, you will need some experience creating and building cases with ACME or CESM. At the minimum, you should be able to determine whether your layout is efficient or not. Improving it will require familiarity with the performance characteristics of individual components.

- Basic input to the process is
  - env_mach_pes.xml file in the case subdirectory. This file defines the component layout.
  - timing/ccsm_timing.${CASE}.${LID} . This file contains the performance data from the run labeled ${LID}. For example for a case named 'ne30_B1850C5' and the LID '150315-130027' this file would be

    ccsm_timing.ne30_B1850C5.150315-130027

    Note that ${LID} is generated by the scripts.
  - (optional) Some general performance goal or requirement.

Small Example: env_mach_pes.xml

- The following fields are a subset of the total, but are the ones relevant to most users. They indicate how many MPI tasks (NTASKS), how many threads per task (NTHRDS), and location of the first task (ROOTPE) for a component (relative to the maximum number of tasks).

```
<entry id="NTASKS_ATM"   value="675"  />
<entry id="NTHRDS_ATM"   value="2"  />
<entry id="ROOTPE_ATM"   value="0"  />

<entry id="NTASKS_LND"   value="168"  />
<entry id="NTHRDS_LND"   value="2"  />
<entry id="ROOTPE_LND"   value="512"  />

<entry id="NTASKS_ICE"   value="512"  />
<entry id="NTHRDS_ICE"   value="2"  />
<entry id="ROOTPE_ICE"   value="0"  />

<entry id="NTASKS_OCN"   value="128"  />
<entry id="NTHRDS_OCN"   value="2"  />
<entry id="ROOTPE_OCN"   value="680"  />
```

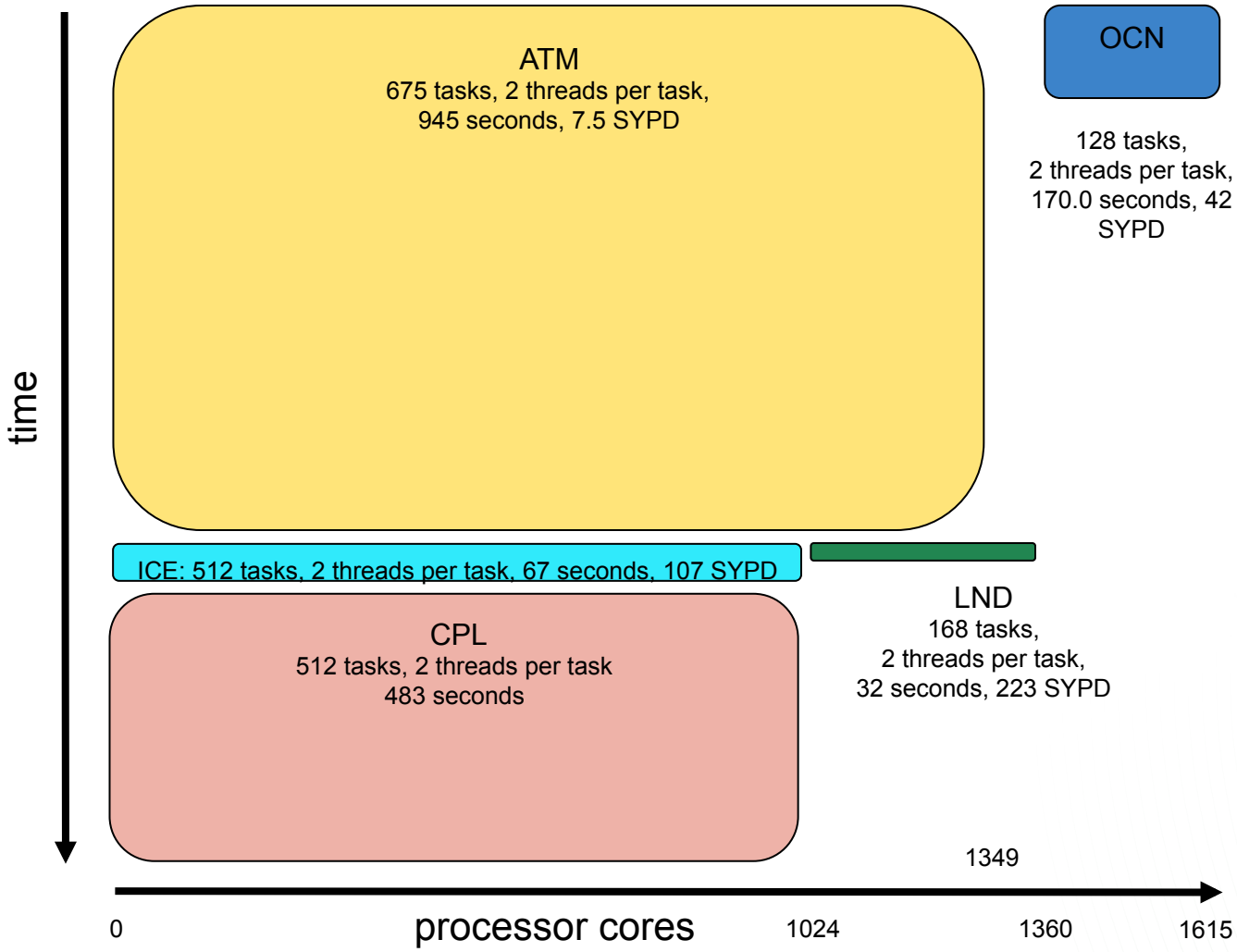
```
<entry id="NTASKS_CPL"   value="512"  />
<entry id="NTHRDS_CPL"   value="2"  />
<entry id="ROOTPE_CPL"   value="0"  />

<entry id="NTASKS_GLC"   value="1"  />
<entry id="NTHRDS_GLC"   value="2"  />
<entry id="ROOTPE_GLC"   value="0"  />

<entry id="NTASKS_ROF"   value="168"  />
<entry id="NTHRDS_ROF"   value="2"  />
<entry id="ROOTPE_ROF"   value="512"  />

<entry id="NTASKS_WAV"   value="512"  />
<entry id="NTHRDS_WAV"   value="2"  />
<entry id="ROOTPE_WAV"   value="0"  />
```

Small Example: ccsm_timing.xxx

- This file has lots of worthwhile performance data and case provenance information, including what component layout was used. For our purposes, looking just at the performance data:

```
grid        : a%ne30np4_l%ne30np4_oi%gx1v6_r%r05_m%gx1v6_g%null_w%null
compset     : 1850_CAM5_CLM40%SP_CICE_POP2_RTM_SGLC_SWAV
run_length  : 30 days (29 for ocean)
…
 Init Time   :      36.677 seconds
 Run Time    :    1117.133 seconds       37.238 seconds/day
…
    TOT Run Time:    1117.133 seconds       37.238 seconds/mday         6.36 myears/wday
    LND Run Time:      31.850 seconds        1.062 seconds/mday       222.96 myears/wday
    ROF Run Time:       1.150 seconds        0.038 seconds/mday      6175.10 myears/wday
    ICE Run Time:      66.940 seconds        2.231 seconds/mday       106.09 myears/wday
    ATM Run Time:     945.307 seconds       31.510 seconds/mday         7.51 myears/wday
    OCN Run Time:     170.009 seconds        5.667 seconds/mday        41.77 myears/wday
    GLC Run Time:       0.000 seconds        0.000 seconds/mday         0.00 myears/wday
    WAV Run Time:       0.000 seconds        0.000 seconds/mday         0.00 myears/wday
    CPL Run Time:     482.530 seconds       16.084 seconds/mday        14.72 myears/wday
    CPL COMM Time:   1068.084 seconds       35.603 seconds/mday         6.65 myears/wday
```

Pictorial Representation



Comments:

- Total cost is 100 seconds more than ATM + ICE (so ~10%), achieving 6.4 SYPD.
- The role of CPL cost in total model cost is something of a mystery (based on these metrics).
- OCN processor cores could could be decreased by ~ a factor 6 without increasing model cost, saving some core hours.
- To speed up model (in this case), need to decrease ATM cost.

Rules of Thumb: Layout

The complexity of the process can be reduced with some general "rules":

- Use the same task decomposition (though not necessarily threads) for
  - ICE and WAV
  - LND and ROF and GLC
- Do not overlap
  - ICE and LND
  - ATM and OCN
- Do overlap
  - ATM and ICE
  - ATM and LND
  - ATM and CPL
- CPL? Use either same task decomposition as LND, as ICE, or as ATM.
- For large problems, may need to run ICE on own processors, for memory reasons.

Rules of Thumb: Interpretation

- Using rules on previous page, total cost is typically a modest percentage over
  - max{ATM + max {ICE, LND}, OCN}
- Minimize core hours by balancing
  - LND and ICE
  - (ATM + max{LND,ICE}) and OCN
- Improve performance by improving ATM and/or ICE.
  - In my recent experience, LND and OCN have not been performance limiters, though that can change with the particulars of the case. This is for simulations with all components active. For OCN-only or LND-only simulations, this will clearly not apply.

Rules of Thumb: Optimization

- Given a performance goal:
  - The first step is to determine a layout for which ATM achieves the goal. (If this is not possible, give up.)
  - The second step is to determine a layout for which ICE achieves the goal.
  - In subsequent steps improve one or the other or both until ATM+ICE achieves the goal. (Much of the CPL cost will be captured by this.)
  - Determine layouts for other components that use processor cores efficiently without violating the performance goal.
- Particulars:
  - An additional talk would be necessary to discuss optimizing ATM layout, and ICE layout, and … Each has different characteristics, though there is nothing magical about this.
  - The runtime performance tuning options referred to earlier are, for the most part, specific to a given component, and also should be considered when optimizing the layout for a given component.
  - Note: I/O can change the performance characteristics, and some of the runtime tuning options relate to I/O.

Large Example: env_mach_pes.xml

- Here ICE is run on its own processor cores, so that it can run all MPI. This improves performance of both ICE (fewer nodes for this MPI task count) and ATM (improved memory access pattern?).

```
<entry id="NTASKS_ATM"   value="21600"  />
<entry id="NTHRDS_ATM"   value="2"  />
<entry id="ROOTPE_ATM"   value="0"  />

<entry id="NTASKS_LND"   value="2048"  />
<entry id="NTHRDS_LND"   value="2"  />
<entry id="ROOTPE_LND"   value="5400"  />

<entry id="NTASKS_ICE"   value="16384"  />
<entry id="NTHRDS_ICE"   value="1"  />
<entry id="ROOTPE_ICE"   value="21600"  />

<entry id="NTASKS_OCN"   value="4166"  />
<entry id="NTHRDS_OCN"   value="2"  />
<entry id="ROOTPE_OCN"   value="37984"  />
```

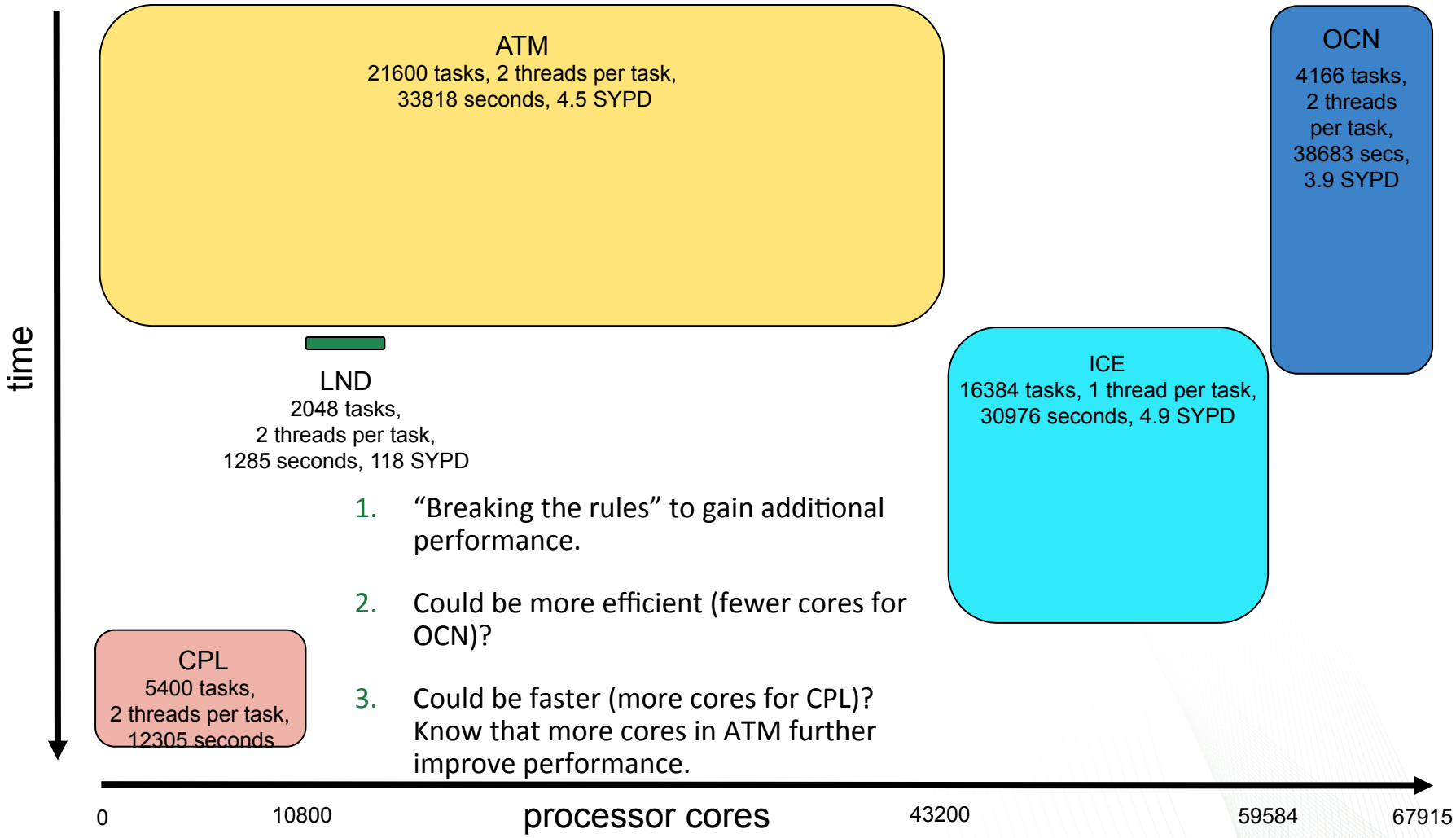
```
<entry id="NTASKS_CPL"   value="5400"  />
<entry id="NTHRDS_CPL"   value="2"  />
<entry id="ROOTPE_CPL"   value="0"  />

<entry id="NTASKS_GLC"   value="3200"  />
<entry id="NTHRDS_GLC"   value="2"  />
<entry id="ROOTPE_GLC"   value="0"  />

<entry id="NTASKS_ROF"   value="1600"  />
<entry id="NTHRDS_ROF"   value="2"  />
<entry id="ROOTPE_ROF"   value="0"  />

<entry id="NTASKS_WAV"   value="3200"  />
<entry id="NTHRDS_WAV"   value="2"  />
<entry id="ROOTPE_WAV"   value="0"  />
```

Large Example: ccsm_timing.XXX

- TOT – (ATM+ICE) = 10,446, which is approximately CPL Run. This layout may make the metric more reliable (or just happenstance)? Difference is 16% of ATM +ICE.

```
grid        : a%ne120np4_l%ne120np4_oi%tx0.1v2_r%r05_m%tx0.1v2_g%null_w%null
compset     : 1850_CAM5_CLM40%SP_CICE_POP2_RTM_SGLC_SWAV
run_length  : 637.999950573349 days (637.999950573349 for ocean)
…
Init Time   :     712.874 seconds
Run Time    :   75239.940 seconds      117.931 seconds/day
…
    TOT Run Time:   75239.940 seconds      117.931 seconds/mday         2.01 myears/wday
    LND Run Time:    1284.818 seconds        2.014 seconds/mday       117.54 myears/wday
    ROF Run Time:      15.667 seconds        0.025 seconds/mday      9639.53 myears/wday
    ICE Run Time:   30976.421 seconds       48.552 seconds/mday         4.88 myears/wday
    ATM Run Time:   33817.703 seconds       53.006 seconds/mday         4.47 myears/wday
    OCN Run Time:   38683.239 seconds       60.632 seconds/mday         3.90 myears/wday
    GLC Run Time:       0.000 seconds        0.000 seconds/mday         0.00 myears/wday
    WAV Run Time:       0.000 seconds        0.000 seconds/mday         0.00 myears/wday
    CPL Run Time:   12304.766 seconds       19.286 seconds/mday        12.27 myears/wday
    CPL COMM Time:  65728.229 seconds      103.022 seconds/mday         2.30 myears/wday
```

Pictorial Representation



1. "Breaking the rules" to gain additional performance.
2. Could be more efficient (fewer cores for OCN)?
3. Could be faster (more cores for CPL)? Know that more cores in ATM further improve performance.

Conclusions

- Earth System Modeling is (often) computationally expensive, and computer performance must be considered to maximize scientific output.
- While the dependencies can be complicated, improving the performance (in terms of runtime or core hours) of individual jobs will typically improve project-wide computational throughput.
- Determining requirements is the first step, both when requesting allocations and when planning computational experiments.
- Improving performance can be nontrivial, and require expert advice. However, certain inefficiencies can be identified easily by users, and addressed directly with training.