Benchmarking using the Community Atmospheric Model

Patrick H. Worley
Oak Ridge National Laboratory

2006 SPEC Benchmark Workshop

January 23, 2006

Joe C. Thompson Conference Center

University of Texas

Austin, Texas

Acknowledgements

- Research sponsored by the Atmospheric and Climate Research Division and the Office of Mathematical, Information, and Computational Sciences, Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.
- These slides have been authored by a contractor of the U.S. Government under contract No. DE-AC05-00OR22725. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes
- Oak Ridge National Laboratory is managed by UT-Battelle, LLC for the United States Department of Energy under Contract No. DE-AC05-00OR22725.

Talk Outline

- Context:
  - Evaluation of Early Systems
  - Performance Portability
- Description of Community Atmospheric Model (CAM)
- Tuning CAM for Benchmarking
- Benchmark Results

Evaluation of Early System (EES)

evaluation study of low serial number or pre-commercial release HPC system with the goal of quickly determining promise of system

- Required Attributes
  - Fast
  - Fair
  - Open
- Hierarchical Approach
  - microbenchmarks for subsystem evaluation
  - kernels for tuning, comparison of programming paradigms, and whole system performance evaluation
  - applications for production performance estimation, generating scaling curves (with respect to processor count and/or problem size)*

* Data useful for price-performance evaluations. We do not address this directly.

Partial Listing of Evaluation Studies at Oak Ridge National Laboratory (ORNL)

1987 - ***Intel iPSC/1***, ***nCUBE 3200***, Ametek S14

1988 - ***Intel iPSC/2***

1990 - ***Intel iPSC/860***, ***nCUBE 6400***

1991 - ***KSR-1,*** Intel Touchstone Delta

1992 - ***Intel Paragon***

1994 - Cray T3D

1995 - ***Intel Paragon MP***, IBM SP2

1996 - Chen-1000, Convex SPP-1200

1998 - SGI Origin2000, Convex SPP-2000, Swiss T0

1999 - ***SRC Prototype***, ***IBM SP (Winterhawk I, Nighthawk I)***

2000 - ***Compaq AlphaServer SC (ES40)***, ***IBM SP (Winterhawk II)***, IBM SP (Nighthawk II)

2001 - ***IBM p690 (with SP Switch2)***, NEC SX-5, HP AlphaServer SC (ES45) , HP AlphaServer GS

2002 - NEC SX-6

2003 - ***Cray X1***, ***SGI Altix 3700***

2004 - ***IBM p690 (with HPS interconnect)***

2005 - ***Cray XT3***, ***Cray XD1***, ***Cray X1E***, SGI Vortex

Years denote when evaluations began (to the best of my recollection). Bold, italicized systems were located at ORNL during the evaluation. Papers describing some of these evaluations are listed in the proceedings paper.

EES and Application Benchmarks

Issue:

Full applications may not take best advantage of the novel features of a new architecture. Kernel results can be used to estimate optimized application performance. However, it is seldom possible to make significant modifications to application codes to validate these estimates within the time frame of an early system evaluation. Modifications may also invalidate comparisons with benchmark data collected on other systems.

Partial Solution:

Use *performance portable* application codes, i.e. application codes that include compile-time and runtime tuning options, targeting a number of different aspects of system performance, that can be used to port and optimize the application on new platforms quickly.

Performance Portability Examples

- Compiler flags, especially support for applying different flags for different routines, and a mechanism to determine solution correctness
- Interfaces to switch between and/or add new math libraries, e.g., for FFT, linear algebra, …
- Memory access optimizations: cache blocking, array alignment, memory stride, …
- Vectorization and software pipelining optimizations: loop lengths, compiler directives for dependency analysis, …
- Interfaces to switch between and/or add new communication layers (MPI, SHMEM, Co-Array Fortran, …), and to switch between different protocols for a given layer
- Parallel programming paradigm options, e.g. MPI, OpenMP, hybrid, …
- Other parallel algorithm options: domain decomposition, load balancing, parallel I/O, process placement, …
- System-specific load options: task migration, memory allocation and migration, small pages vs. large pages, process placement, …
- …

EES and Performance Portable Benchmarks

Issue:

Performance portable codes support fairer benchmarking without requiring code modifications that might invalidate comparisons with earlier benchmark data. However, interplatform comparisons (still) require that similar effort be expended on all platforms. Unfortunately, the more options to be explored, and the “fairer” we attempt to be, the more expensive (and slow) the benchmarking process becomes.

Partial Solution:

Specify a methodology that minimizes the expense of tuning for benchmark runs and that can be applied in all evaluation exercises.

Community Atmospheric Model (CAM)

Atmospheric global circulation model

- Timestepping code with two primary phases per timestep
  - *Dynamics*: advances evolution equations for atmospheric flow
  - *Physics*: approximates subgrid phenomena, such as precipitation, clouds, radiation, turbulent mixing, …
- Multiple options for dynamics:
  - Spectral Eulerian (EUL) dynamical core (*dycore*)
  - Spectral semi-Lagrangian (SLD) dycore
  - Finite-Volume semi-Lagrangian (FV) dycore

  all using tensor product *longitude* x *latitude* x *vertical level* grid over the sphere, but not same grid, same placement of variables on grid, or same domain decomposition in parallel implementation.
- Separate data structures for dynamics and physics and explicit data movement between them each timestep (in a "coupler")
- Developed at NCAR, with contributions from DOE and NASA
- http://www.ccsm.ucar.edu/models/atm-cam/

Source Material

P. Worley and J. Drake, *Performance Portability in the Physical Parameterizations of the Community Atmospheric Model*, International Journal for High Performance Computer Applications, 19(3), August 2005, pp. 187-201.

A. Mirin and W. Sawyer, *A Scalable Implementation of a Finite-Volume Dynamical Core in the Community Atmosphere Model*, International Journal for High Performance Computer Applications, 19(3), August 2005, pp. 203-212.

W. Putman, S-J. Lin, and B-W. Shen, *Cross-Platform Performance of a Portable Communication Module and the NASA Finite Volume General Circulation Model*, International Journal for High Performance Computer Applications, 19(3), August 2005, pp. 213-223.

L. Oliker, J. Carter, M. Wehner, A. Canning, S. Ethier, A. Mirin, G. Bala, D. Parks, P. Worley, S. Kitawaki, and Y. Tsuda, *Leading Computational Methods on Scalar and Vector HEC Platforms*, in Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking and Storage (SC05), Seattle, WA, November 12-18, 2005.

CAM Performance Portability Goals

1) Maximize single processor performance, e.g.
   a) Optimize memory access patterns
   b) Maximize vectorization or other fine-grain parallelism
2) Minimize parallel overhead, e.g.
   a) Minimize communication costs
   b) Minimize load imbalance
   c) Minimize redundant computation

for

- a range of target systems,
- a range of problem specifications (grid size, physical processes, …)
- a range of processor counts

while preserving maintainability and extensibility.

No optimal solution for all desired (platform,problem,processor count) specifications. Approach: compile-time and runtime optimization options.

Performance Optimization Options

- Physics data structures
  - Index range, dimension declaration
- Physics load balance
  - Variety of load balancing options, with different communication overheads
  - SMP-aware load balancing options
- Communication options
  - MPI protocols (two-sided and one-sided)
  - Co-Array Fortran
  - SHMEM protocols

  and pt-2-pt implementations or collective communication operators
- OpenMP parallelism
  - Instead of some MPI parallelism
  - In addition to MPI parallelism
- Aspect ratio of dynamics 2D domain decomposition (FV-only)

Physics Data Structures

- All three dynamics use a tensor product longitude-vertical-latitude (plon x pver x plat) computational grid covering the sphere.
- A *vertical column* is a set of grid points of coordinates (i,*,j). In current physics, computation is independent between vertical columns, and tightly coupled within a vertical column.
- The basic data structure in the physics is the *chunk*, an arbitrary collection of vertical columns. Grid points in a chunk are referenced by (local column index, vertical index).
- Define

  ncols(j): number of columns allocated to chunk j

  nchunks: number of chunks

  begchunk:endchunk: chunk indices assigned to a given process

  pcols: maximum number of columns allocated to any chunk

  pcols and pver specified at compile time.
- Arrays declared as (pcols, pver, begchunk:endchunk).

Physics Computational Structure

- Loops structured as

```
do j=begchunk,endchunk
   do k=1,pver
      do i=1,ncols(j)
         (physical parameterizations)
      enddo
   enddo
enddo
```

- Inner loop over columns is vectorizable.
- Coarser grain parallelism is exploited over outer loop over chunks, OpenMP can be used to parallize j loop more.
- Length of inner loop can be adjusted for size of cache or for vector length.
- Columns can be assigned to chunk in order to balance load between chunks or to minimize communication cost of coupling between dynamics and physics.

Benchmarking Tuning Methodology

- Use microbenchmarks or kernels to determine small subset of communication protocols and messaging layers to use in later benchmarking.
- Use small number of full applications runs to determine optimal compiler settings, math libraries, load-time environment variable settings (may result in small subset)
- Tune `pcols` on a small number of processors, varying problem size to examine variety of granularities
- When benchmarking, optimize performance over *candidate* tuning options, intelligently pruning search as vary processor count:
  - Time the code when simulating one, two, and three simulation days, taking differences to eliminate atypical start-up costs. Only run longer simulations for options achieving best performance for one simulation day.
  - Look for “crossover points” in processor count or problem size , then use subsets of tuning options within indicated intervals.

Platforms

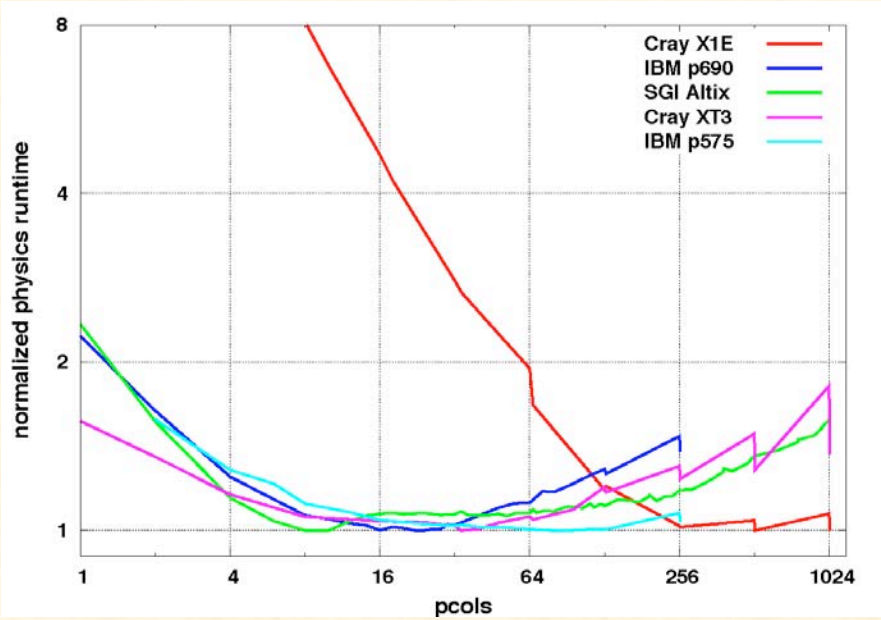
- **Cray X1** at ORNL: 128 4-way vector SMP nodes and a 4-D hypercube interconnect. Each processor has 8 64-bit floating point vector units running at 800 MHz.
- **Cray X1E** at ORNL: 256 4-way vector SMP nodes and a 4-D hypercube interconnect. Each processor has 8 64-bit floating point vector units running at 1.13 GHz.
- **Cray XT3** at ORNL: 5294 single processor nodes (2.4 GHz AMD Opteron) and a 3-D torus interconnect.
- **Earth Simulator**: 640 8-way vector SMP nodes and a 640x640 single-stage crossbar interconnect. Each processor has 8 64-bit floating point vector units running at 500 MHz.
- **IBM p575 cluster** at the National Energy Research Supercomputer Center (NERSC): 122 8-way p575 SMP nodes (1.9 GHz POWER5) and an HPS interconnect with 1 two-link network adapter per node.
- **IBM p690 cluster** at ORNL: 27 32-way p690 SMP nodes (1.3 GHz POWER4) and an HPS interconnect with 2 two-link network adapters per node.
- **IBM SP** at NERSC: 184 Nighthawk II 16-way SMP nodes (375MHz POWER3-II) and an SP Switch2 with two network adapters per node.
- **Itanium2 cluster** at Lawrence Livermore National Laboratory (LLNL): 1024 4-way Tiger4 nodes (1.4 GHz Intel Itanium 2) and a Quadrics QsNetII Elan4 interconnect.
- **SGI Altix 3700** at ORNL: 2 128-way SMP nodes and NUMAflex fat-tree interconnect. Each processor is a 1.5 GHz Itanium 2 with a 6 MB L3 cache.

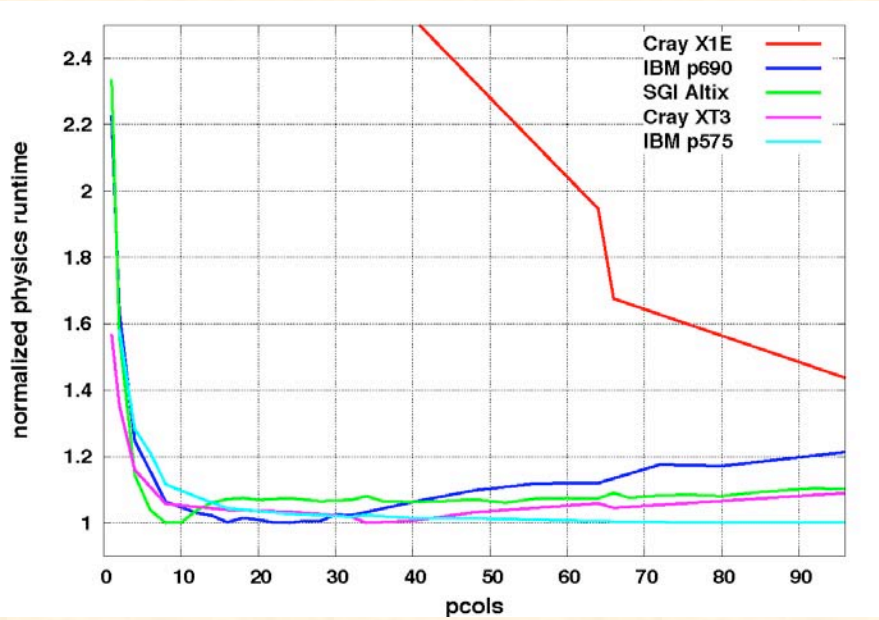
`pcols` Tuning Experiments

- "Single" processor performance tuning, using MPI only and
  - all processors in SMP node, to include impact of memory contention, or
  - two processors for systems with single processor nodes

  solving problems with
  - 2048 columns (26 vertical levels, 64 x 32 horizontal grid), so 1024 columns per processor in two processor experiments, or
  - 32768 columns (26 vertical levels, 256 x 128 horizontal grid)
- Columns assigned to chunks both "in order" and to balance load between chunks. (Results similar in both cases.)
- CAM executed for one simulation day and for two simulation days. Difference examined to check for atypical start-up costs.
- Varied `pcols` (which necessarily varied `ncols(j)`).
- Physics-only execution times for all processes summed, and results for each `pcols` value normalized with respect to minimum observed time over all experiments for a given platform.

pcols Tuning Experiments





Altix: minimum at pcols = 8

p690: minimum at pcols = 24

XT3: minimum at pcols = 34

p575 : minimum at pcols = 80

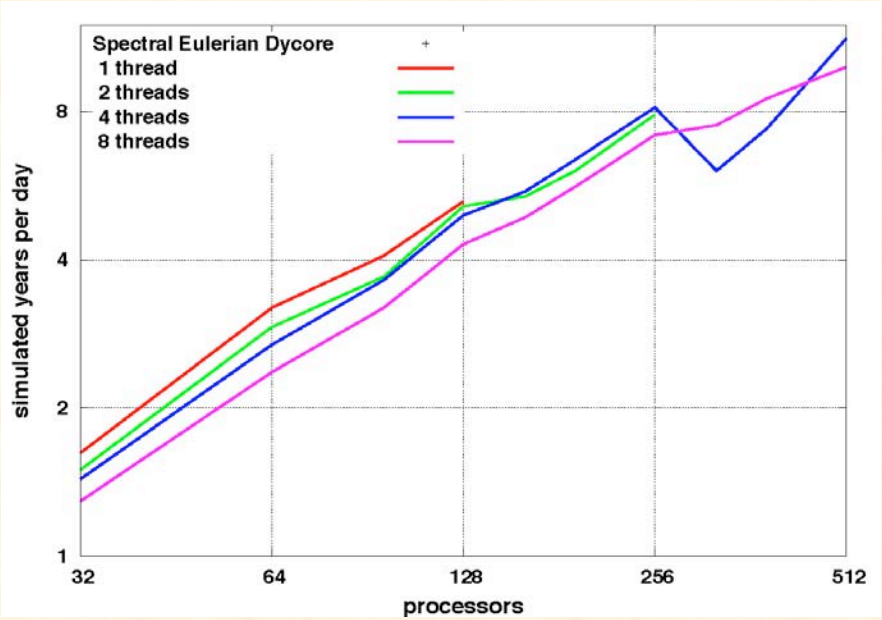
X1E : minimum at pcols = 514 or 1026

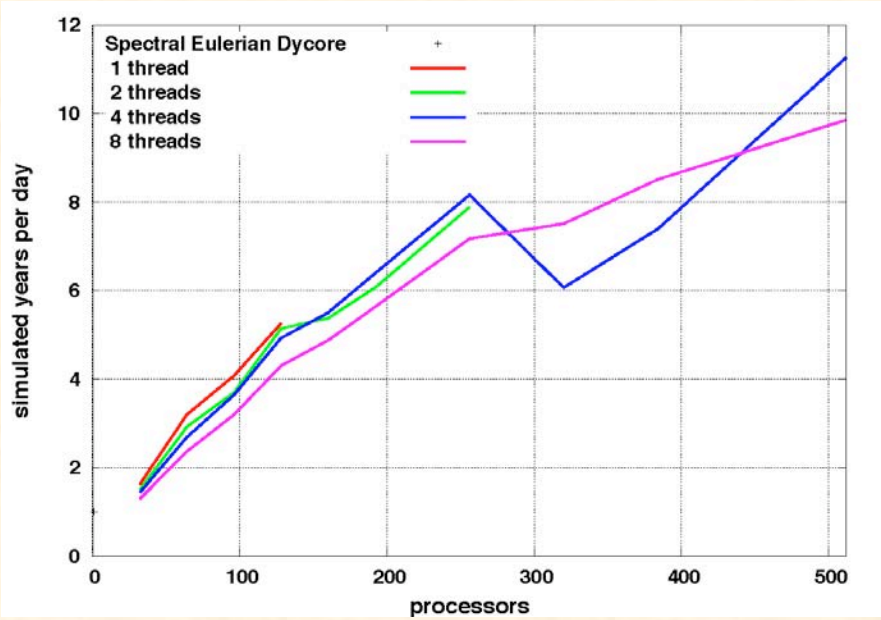
pcols <= 4 bad for all systems.

MPI vs. OpenMP

- Parallel performance as a function of processor count for different numbers of OpenMP threads per process
- Total runtime for a typical simulation day measured on IBM p690 cluster and used to calculate computation rate in simulation years per wallclock day.
- Two dycores examined:
  - spectral Eulerian dycore solving problem with 256 x 128 horizontal grid and 26 vertical levels
  - finite volume dycore solving problem with 576 x 381 horizontal grid and 26 vertical levels
- For finite volume dycore, examined both one dimensional over latitude and two dimensional decompositions. Two dimensional decomposition used (P/4)x4 virtual processor grid where first dimension decomposes latitude dimension.
- Performance results optimized over communication protocol, `pcols`, and load balancing (separately for each data point).

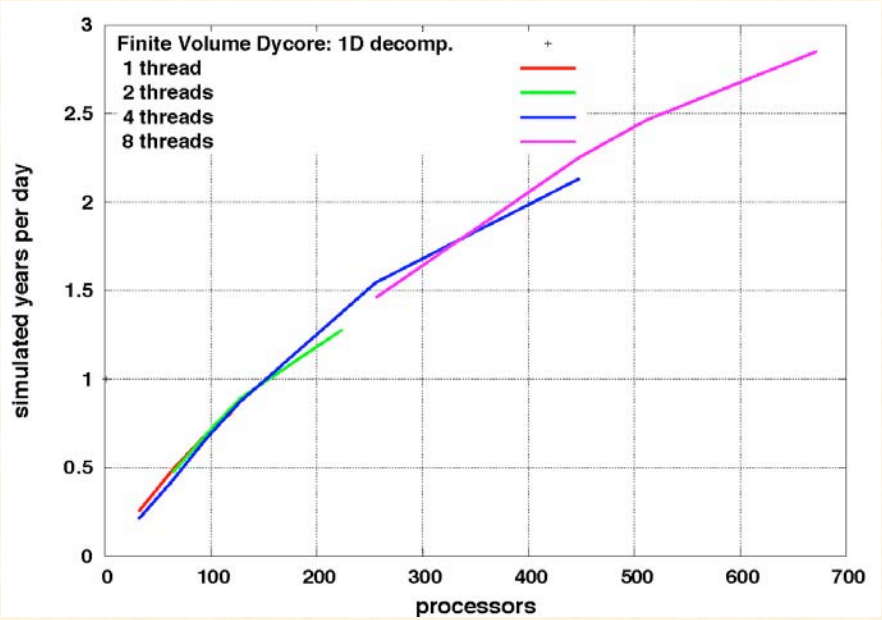
MPI vs. OpenMP: Spectral Eulerian Dycore

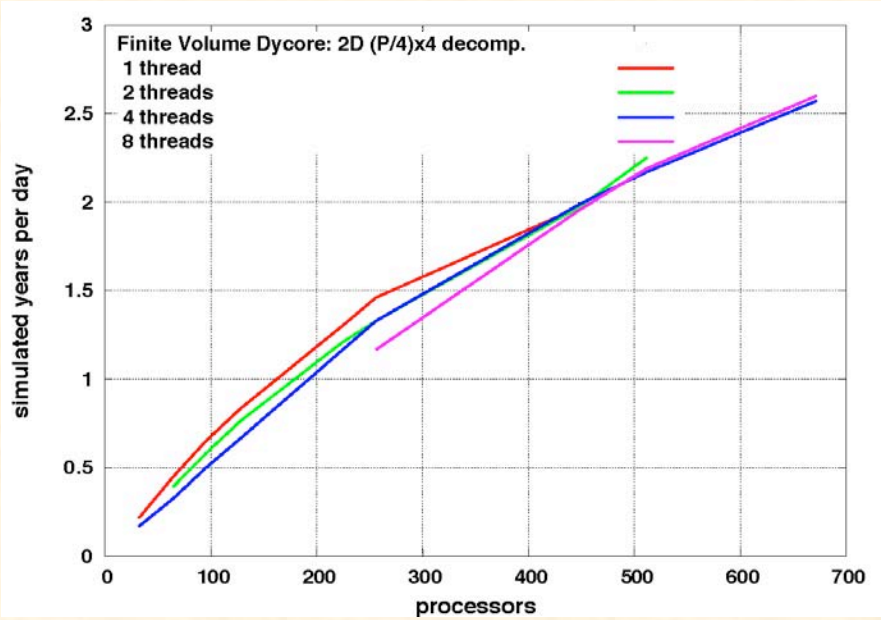




- Primary utility of OpenMP is to extend scalability.
- In general, fewer threads is better for the same number of processors
- Exceptions to previous observation can occur, and choice of number of threads should be determined on a case-by-case basis.

MPI vs. OpenMP: Finite Volume Dycore





- Primary utility of OpenMP is, again, to extend scalability.
- For 2D decomposition, fewer threads is generally better for the same number of processors.
- For 1D decomposition, it is more important not to let the number of processors decomposing latitude exceed approx. 64 than it is to minimize the number of threads.

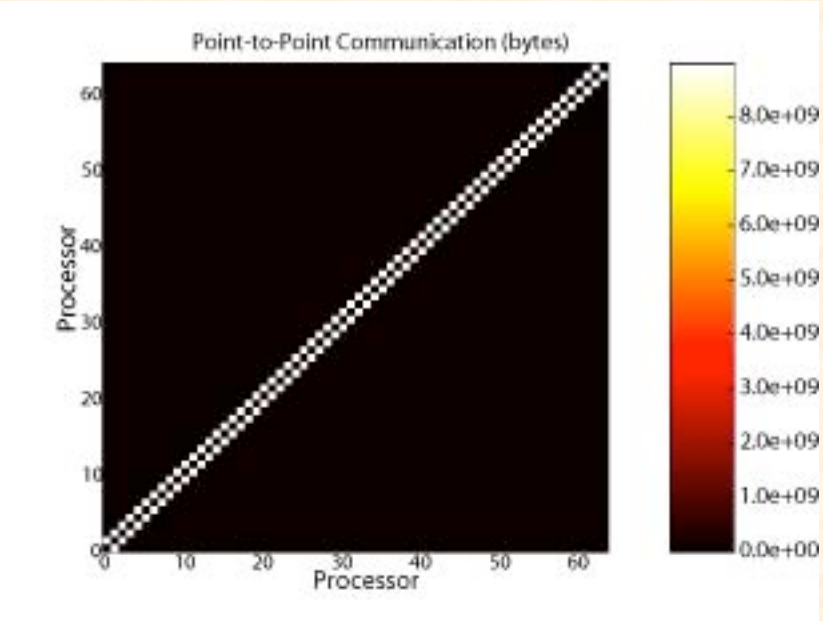
1D vs. 2D Decompositions

- Parallel performance as a function of processor count for different domain decompositions for finite volume dycore:
  - 1D over latitude
  - 2D, defined by a (P/4)x4 virtual processor grid
  - 2D, defined by a (P/7)x7 virtual processor grid

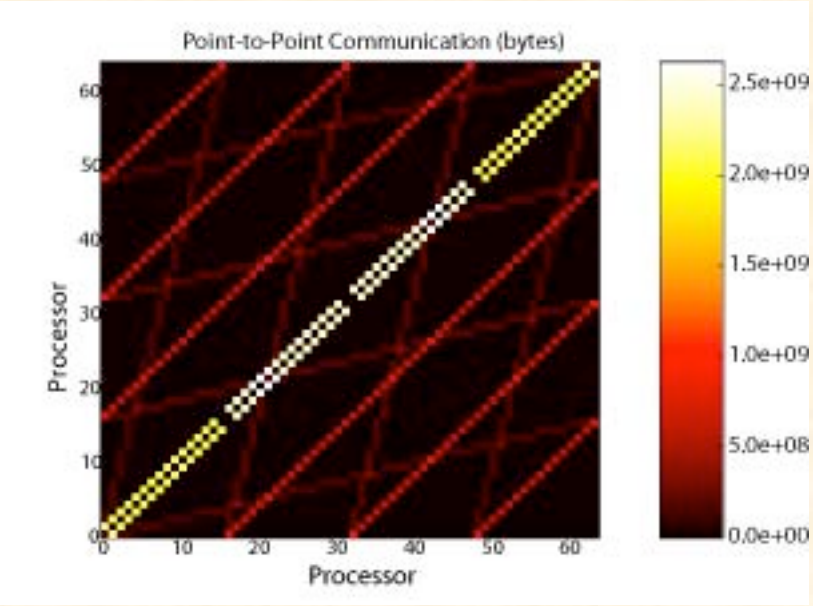
  where first dimension in the processor grid decomposes latitude dimension
- Total runtime for a typical simulation day, measured on
  - Cray X1E
  - Cray XT3
  - IBM p690 cluster

  and used to calculate computation rates.
- Performance results optimized over communication protocol, `pcols`, load balancing, and number of OpenMP threads (separately for each data point).

Communication: Finite Volume Dycore



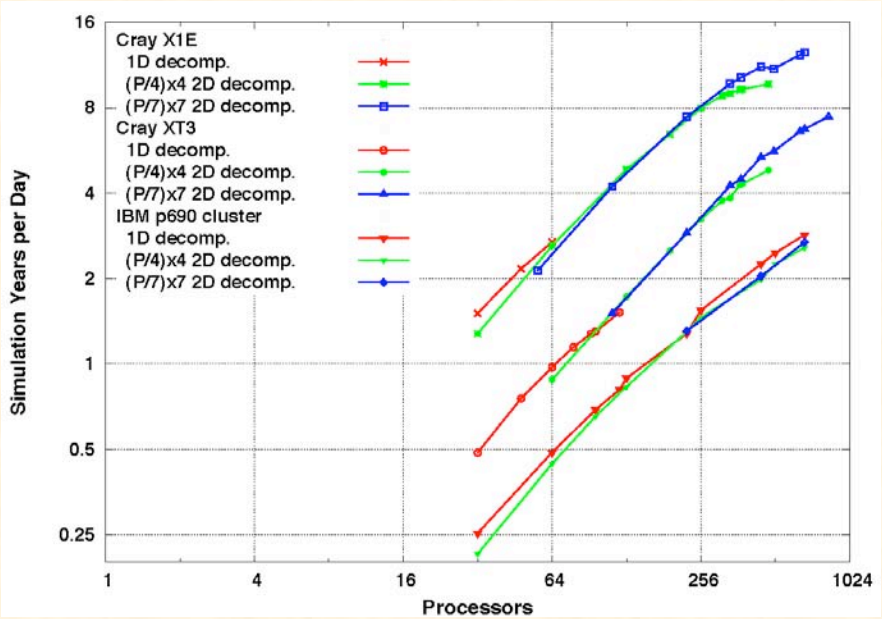
1D Decomposition
64x1

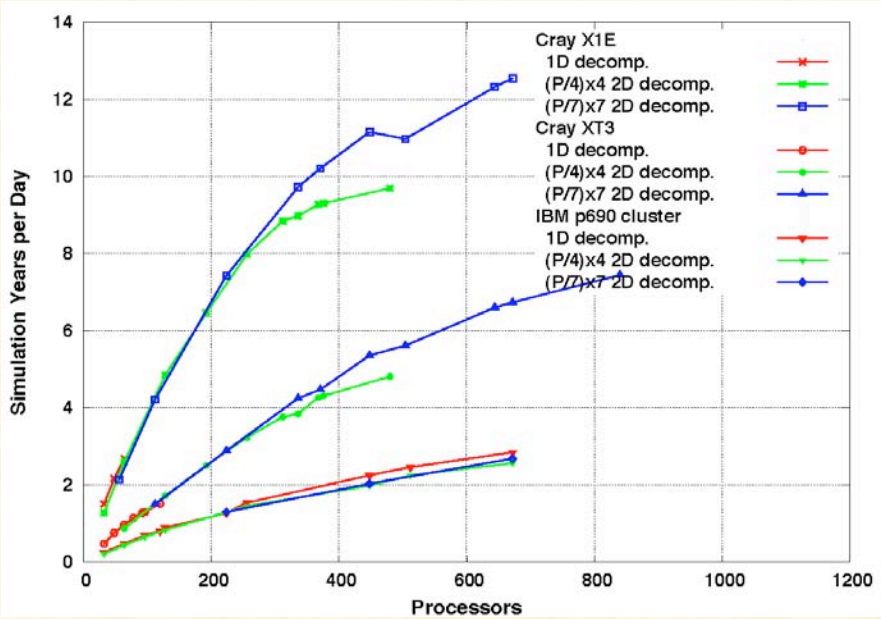


2D Decomposition
16x4

(from L. Oliker, et al, *Leading Computational Methods on Scalar and Vector HEC Platforms,* in SC05 Proceedings.)

1D vs. 2D: Finite Volume Dycore





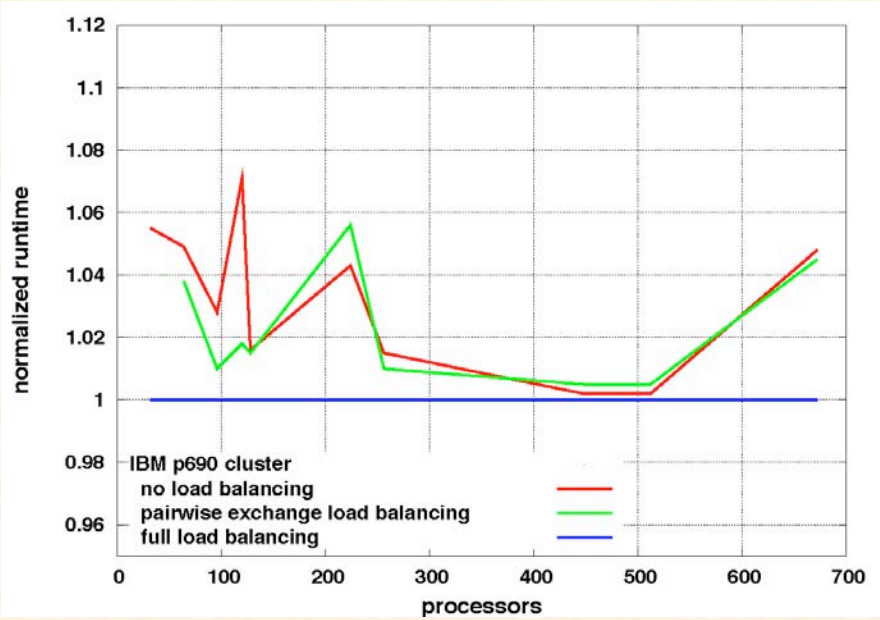
- 2D decompositions become superior to 1D when number of MPI processes used to decompose latitude dimension in 1D exceeds some limit, ~70 on two Cray systems (not using OpenMP). Similarly, (P/7)x7 decomposition begins to outperform (P/4)x4 when the number of MPI processes decomposing latitude in (P/4)x4 exceeds ~70.
- On IBM p690, using OpenMP and the 1D decomposition is superior to using the 2D decomposition up to 672 processors, even when the optimum for 1D uses more threads per process than the optimum for 2D. Note 1D never uses more than 84 MPI processes.

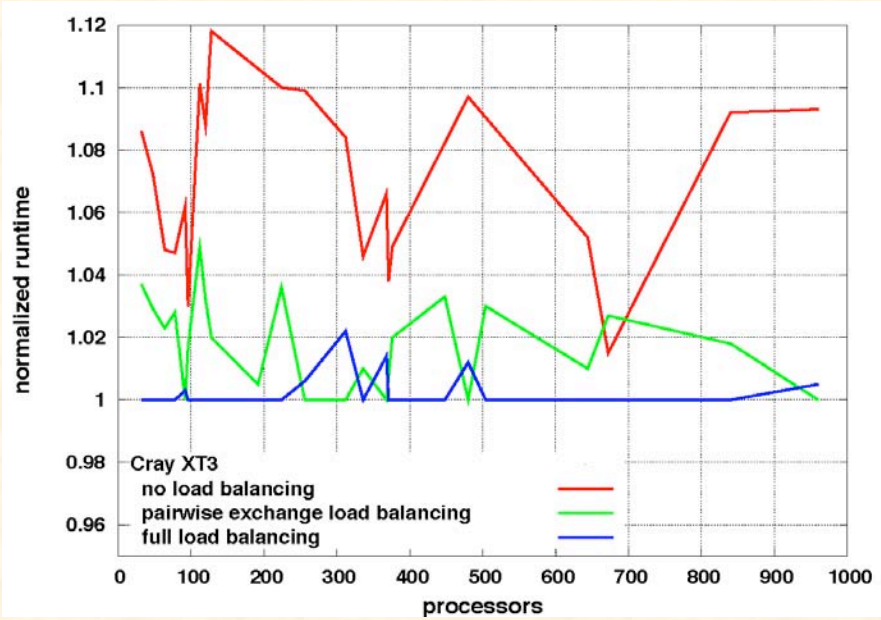
Load Balancing Comparisons

- Parallel performance as a function of processor count for different load balancing schemes for finite volume dycore:
  - Load balancing only within MPI process (no interprocess communication)
  - Load balancing only between pairs of processes (single step, pairwise interprocess communicaiton)
  - Best load balancing (requiring MPI_Alltoallv functionality)
- Total runtime for a typical simulation day, measured on
  - Cray XT3
  - IBM p690 cluster

  and divided by the runtime for the minimum over all load balancing schemes for a given processor .
- Performance results optimized over communication protocol, `pcols`, domain decomposition, and number of OpenMP threads (separately for each data point).

Load Balancing: Finite Volume Dycore



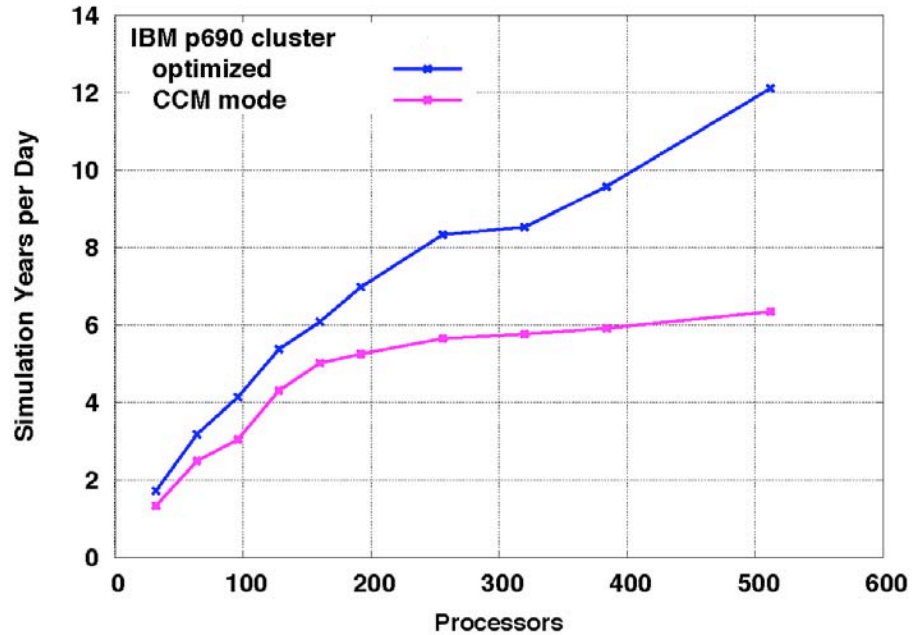


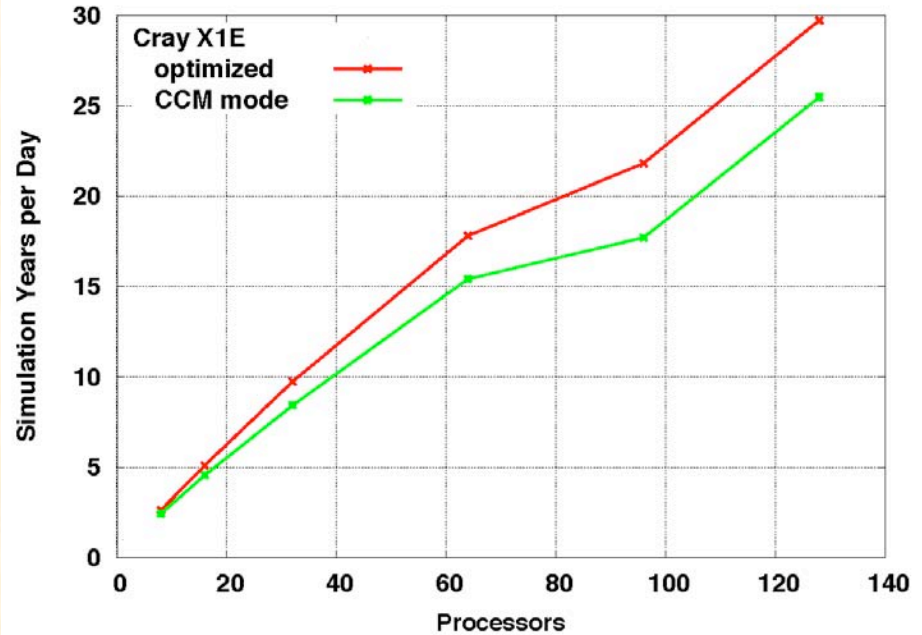
- On the IBM p690, full load balancing is always best, but no load balancing is, at worst, only 7% slower. (For spectral dycore, full load balancing not always optimal.)
- On the Cray XT3, full load balancing is usually best, but pairwise exchange load balancing is competitive. No load balancing is usually more than 4% slower, and as much as 12% slower.
- The best example of the advantage of full load balancing is on the Cray X1E. It is so much better that it was clear early on in the tuning process and the other load balancing options were “pruned” from the search tree.

Benchmarks: Spectral Eulerian Dycore

- 256 x 128 x 26 problem
  - CAM resolution when used in Community Climate System Model production runs.
- Optimized over `pcols`, load balancing options, number of OpenMP threads per process for a given total number of processors, and communication protocols for each system and processor count.
- Also run in "original" formulation, `pcols=256`, `258`, and `259` with no load balancing. Still optimized over number of OpenMP threads per process. Referred to as "CCM" settings, where CCM is the CAM predecessor.
- Optimal setting determined using one and two simulation day experiments.
- Benchmark runs are 30 simulation days in September .

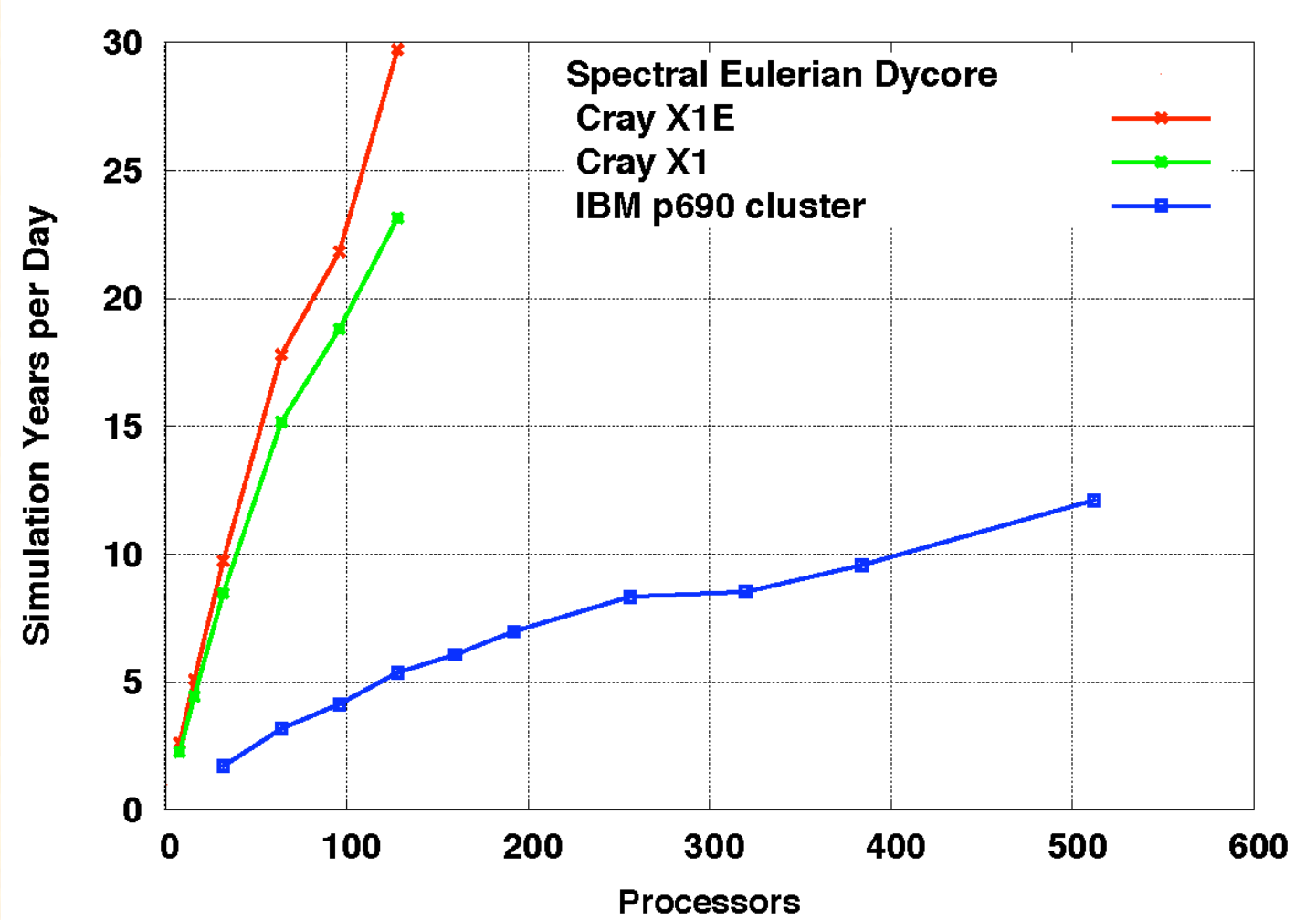
Tuning Impact: Spectral Eulerian Dynamics





- Big win on IBM, primarily due to ability to use OpenMP parallelism in physics when dynamics parallelism exhausted (at approximately 128 processors).
- As vector length for CCM mode is near optimal on the X1E, performance difference is primarily load balancing.

Benchmarking: Spectral Eulerian Dycore



Benchmarking Details: IBM p690 cluster

Proc.	MPI proc.s	OpenMP threads per proc.	pcols	load balance	MPI collective	improv. vs. CCM
32	32	1	16	full	yes	28%
64	64	1	16	pair	no	28%
96	96	1	24	full	yes	36%
128	128	1	16	pair	no	25%
160	40	4	24	full	yes	21%
192	48	4	24	pair	no	33%
256	128	2	16	pair	no	47%
320	40	8	32	none	-	48%
384	48	8	24	pair	no	62%
512	128	4	16	pair	no	91%

Benchmarking Details: Cray X1

Proc.	MPI proc.s	OpenMP threads per proc.	pcols	load balance	MPI collective	improv. vs. CCM
8	8	1	1026	full	yes	19%
16	16	1	1026	full	yes	23%
32	32	1	1026	full	yes	24%
64	64	1	514	full	yes	21%
96	96	1	514	full	yes	30%
128	128	1	258	full	yes	10%

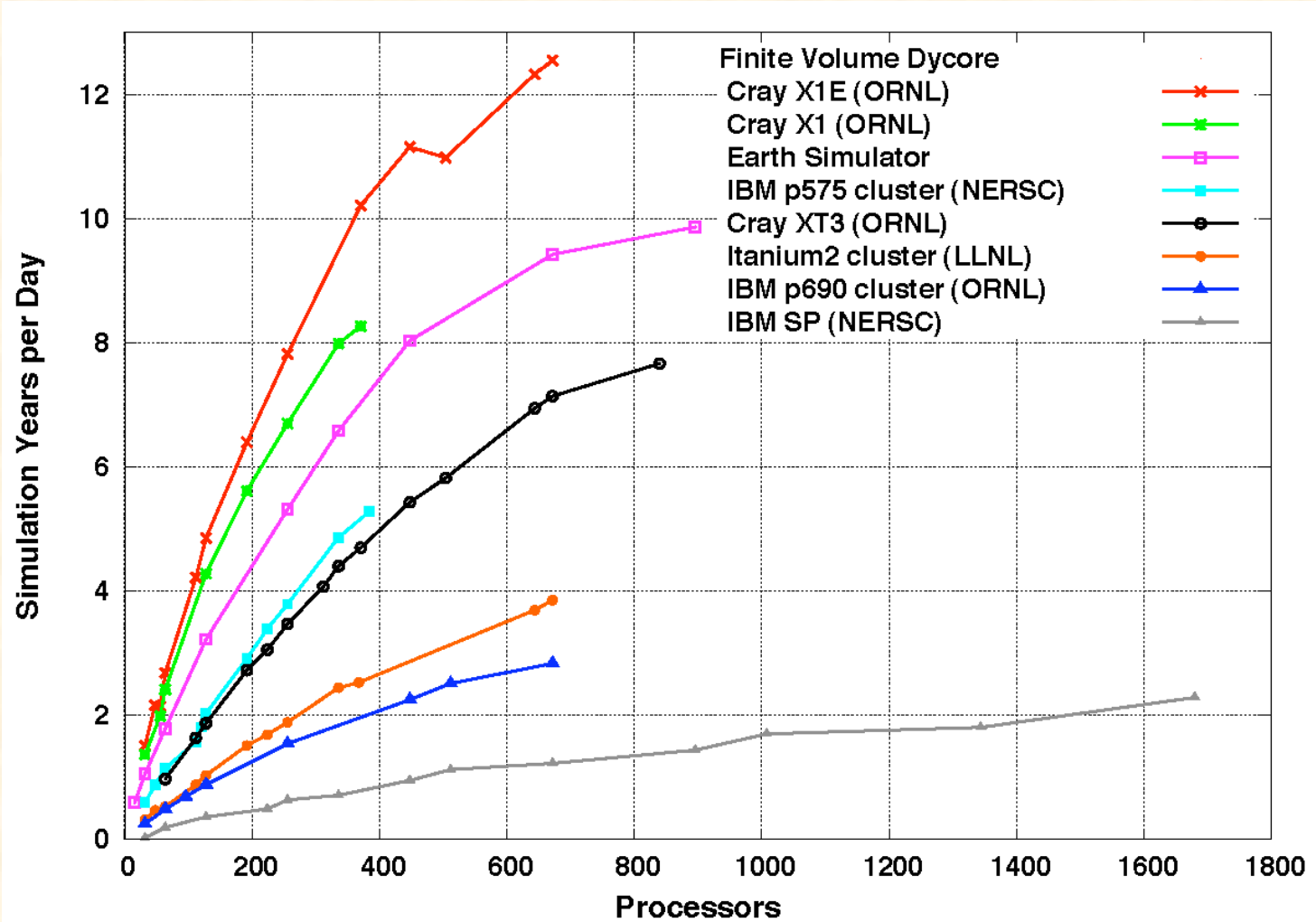
Benchmarking Details: Cray X1E

Proc.	MPI proc.s	OpenMP threads per proc.	pcols	load balance	MPI collective	improv. vs. CCM
8	8	1	1026	full	yes	8%
16	16	1	1026	full	yes	12%
32	32	1	1026	full	yes	16%
64	64	1	514	full	yes	15%
96	96	1	514	full	yes	23%
128	128	1	258	full	yes	17%

Benchmarks: Finite Volume Dycore

- 576 x 381 x 26 problem
- Optimized over `pcols`, virtual process grid defining domain decomposition, load balancing options, number of OpenMP threads per process for a given total number of processors, and communication protocols for each system and processor count (including MPI collectives for load balancing vs point-to-point implementations and communicating into/out of derived types for dycore communication).
- No reasonable "default" tuning options available for comparison with optimized performance - simply reporting performance.
- Optimal setting determined using one and three simulation day experiments. Results calculated by (runtime for 3 sim. days - runtime for 1 sim. day)/2, and reported as simulation years per day.
- p575 cluster results are without OpenMP; experiments are ongoing.
- Results for Earth Simulator, IBM SP, and Itanium2 cluster collected by others with guidance as to tuning methodology. Some benchmark details are not available.

Benchmarking: Finite Volume Dycore



Benchmarking Details: IBM p690 cluster

Proc.	process grid	OpenMP threads per proc.	pcols	load balance	MPI collective for load bal.	MPI with derived types
32	32x1	1	16	full	yes	yes
64	64x1	1	16	full	yes	yes
96	48x1	2	16	full	yes	yes
128	64x1	2	16	full	yes	yes
224	32x7	1	24	full	yes	yes
256	64x1	4	24	full	yes	yes
448	56x1	8	32	full	yes	yes
512	64x1	8	32	full	yes	yes
672	84x1	8	32	full	yes	yes

Benchmarking Details: Cray XT3

Proc.	process grid	OpenMP threads per proc.	pcols	load balance	MPI collective for load bal.	MPI with derived types
32	32x1	1	40	full	yes	yes
48	48x1	1	40	full	yes	no
64	64x1	1	40	full	yes	yes
96	24x4	1	40	full	yes	yes
128	32x4	1	40	full	yes	yes
192	48x4	1	40	full	yes	yes
256	64x4	1	40	full	yes	yes
336	48x7	1	40	full	yes	no
448	64x7	1	40	full	yes	yes
672	96x7	1	40	full	yes	yes

Benchmarking Details: Cray X1

Proc.	process grid	OpenMP threads per proc.	pcols	load balance	MPI collective for load bal.	MPI with derived types
32	32x1	1	870	full	yes	no
64	16x4	1	870	full	yes	no
128	32x4	1	870	full	yes	no
192	48x4	1	990	full	yes	no
256	64x4	1	870	full	yes	no
336	48x7	1	630	full	yes	no
371	53x7	1	570	full	yes	no

Benchmarking Details: Cray X1E

Proc.	process grid	OpenMP threads per proc.	pcols	load balance	MPI collective for load bal.	MPI with derived types
32	32x1	1	870	full	yes	no
48	48x1	1	870	full	yes	no
64	64x1	1	870	full	yes	no
96	24x4	1	870	full	yes	no
128	32x4	1	870	full	yes	no
192	48x4	1	990	full	yes	no
256	64x4	1	870	full	yes	no
336	48x7	1	630	full	yes	no
448	64x7	1	570	full	yes	no
672	96x7	1	330	full	yes	no

Restatement of Thesis

- It can be difficult to use and to interpret results of application benchmarks when evaluating early systems.
- Application codes that have been engineered to be performance portable are easier to port and to optimize, and produce more meaningful results.
- Achieving fairness can be expensive even for performance portable applications, but a careful tuning and benchmarking methodology can minimize this cost.