Performance Optimization at Scale

Recent Experiences

Patrick H. Worley
Oak Ridge National Laboratory

Workshop on Performance Analysis of Extreme-Scale Systems and Applications
Los Alamos Computer Science Symposium
October 15, 2008
La Fonda Hotel
Santa Fe, New Mexico

Acknowledgements

- The work described in this presentation was sponsored by the Atmospheric and Climate Research Division, the Fusion Energy Sciences Program, and the Office of Mathematical, Information, and Computational Sciences, all of the 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 with UT-Battelle, LLC. 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.
- This work used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725, of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract AC02-06CH1135, and of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Recent Activities

1. Application performance engineering and optimization targeting the tera-, peta-, and exa-scale:
   a. XGC-1 gyrokinetic turbulence "edge" code
   b. Community Atmosphere Model (CAM)

   focusing in particular on parallel algorithm design, evaluation, and implementation (both MPI and OpenMP).
2. Performance evaluation of prototype petascale HPC systems:
   a. IBM BG/P
   b. Cray XT4 (quad-core)

As both performance optimization and performance evaluation are important customers of performance analysis, my activities may be of interest to this audience.

Process

- For application code optimization, profile data (timers and PAPI counters) are collected and used to guide subsequent empirical experiments. Focus is less on “mining” performance data from a single run, and more on comparing data collected across multiple runs. Data may come from runs with different settings of existing runtime options or from manually modifying the code to, for example, better characterize a performance issue or evaluate an alternative approach.
- For performance evaluation, microbenchmarks are used to characterize subsystem performance. These characterizations are then used to define and to interpret empirical experiments used in application benchmarking.

Process Characteristics

1. Performance analysis and optimization “from the inside”, not treating a code as a black box
2. Targeted experiments, collecting data to examine specific issues and modifying the code as a natural part of the process
3. Many (short) experiments, as work through possible problems and possible solutions
4. Identifying issues at large scale, but not working at scale except when necessary

Questions

- Is this process feasible at scale?
- Is it more or less feasible than a less hands-on approach to identify and address performance issues?
- Are there alternatives that make more sense at scale?

Practical Issues

1. Need benchmarks representative of goals of work, both code versions and problem specifications
2. Need interactive sessions and/or fast turnaround for batch requests
3. Need sufficient and predictable access to required computing resources, including large processor count runs
4. Would like additional support for controlled experiments:
   a. Information on and/or better control of environment (including system software versions and default environment variables)
   b. Support for requesting specific configurations
   c. Global system performance data: where am I running, where are others running, what are they doing, and what shared resources are we competing for.

Most of these are NOT technical issues, involving instead adequate support from application teams and from computing centers.

Recent Performance Issues

1. System Limitations, e.g.
   a. Limit on number and volume of "unexpected messages" when using MPI on the XT4
   b. Limit on number of MPI subcommunicators on BG/P
   c. Poor performance from MPI collectives, especially at scale

   Note that all of these can be addressed via algorithm modifications.
2. Performance variability due to contention with other users
   a. Intrinsic system hotspots (e.g., file system)?
   b. Way system is run (e.g., allocation policy)?

   More difficult to address? Simply try to recognize problem and try again later? Complain?
3. System failures (often beginning with degraded performance).

Recent Performance Issues

4. Explicitly unscalable algorithms, e.g.
   a. single reader/writer I/O
   b. (depending on file system) every process reads and writes
   c. master-controlled diagnostics
   d. undistributed data structures (replicated for “convenience”), and associated algorithms required to maintain the data structure
5. Implicitly unscalable algorithms, e.g.
   a. certain types of load imbalances
   b. communication-intensive parallelization strategies, e.g. transpose-based algorithms

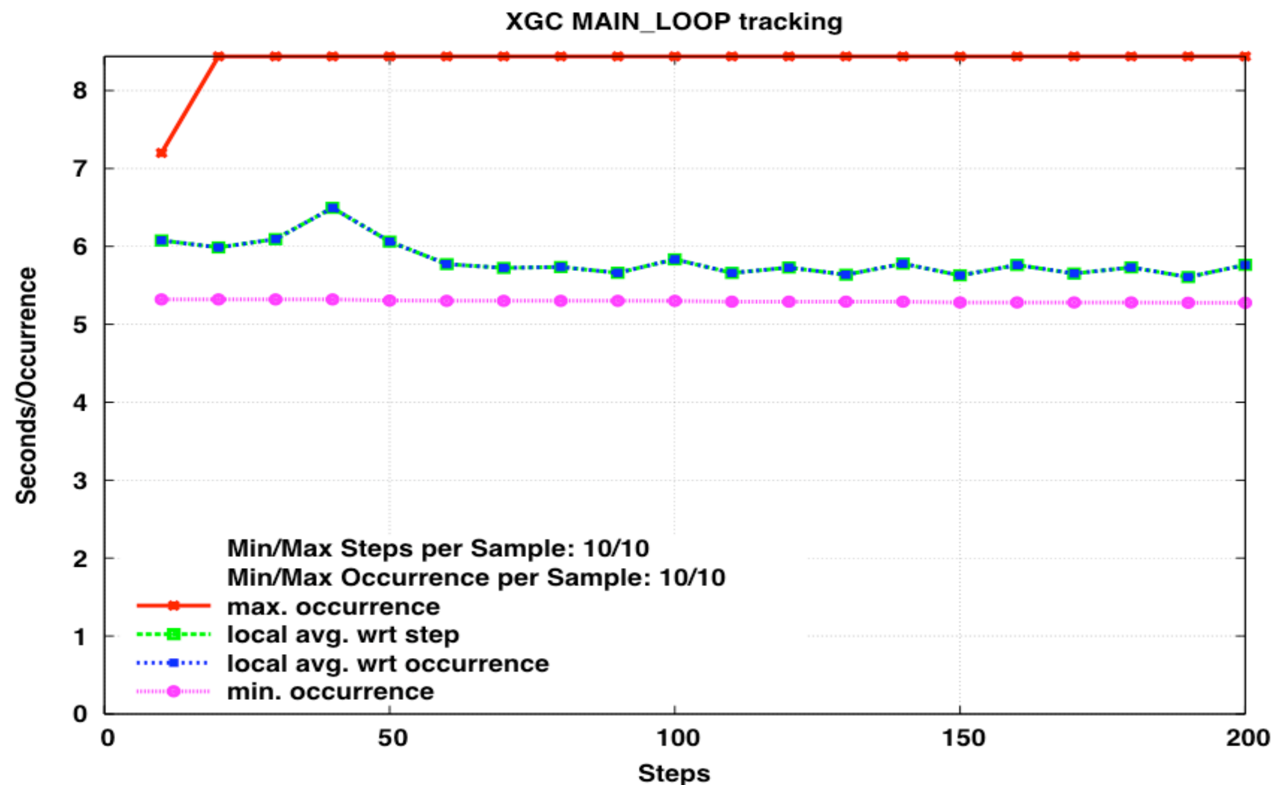
4. and 5. are typically easy to diagnosis, but can be difficult to address.

OAK RIDGE National Laboratory

Office of Science U.S. DEPARTMENT OF ENERGY

XGC-1

1. Introduced logic to dump profile data periodically during run and visualize it in real-time or post-mortem, in order to identify performance variability.
   - Seemed like a good idea at the time. Its utility has yet to be proven as it is not being used in production runs yet.



OAK RIDGE National Laboratory

Office of Science U.S. DEPARTMENT OF ENERGY

XGC-1

2. Requested dedicated time to run experiments at scale (20,000 processors) on the XT4 to provide performance data for use in a proposal, with a last minute twist:

   "I tried 16k cores a few times and they all crashed. 8k cores are fine. … If you get the emergency reservation, and the code doesn't work, please try to debug it." (from code developer)

   - The runs did abort when using 16K and 20K processes. I "quickly" tried 7 different modifications of MPI logic, each of which worked but with different performance. I didn't achieve performance comparable to the original logic until the next day. Was given higher priority for a noninteractive run to collect final data.

XGC-1: MPI problem

The XGC-1 experiments were a mix of weak and strong scaling. XGC-1 is a particle-in-cell code. The underlying grid size was fixed independent of the number of processors (strong scaling) and the number of particles assigned to each processor was fixed (weak scaling). The routine where the code was dying (SHIFT) identifies and moves particles that had left the regions assigned to the particular process. The original logic was

1. determine where to send particles (allreduce + point-to-point)
2. send all particles that need to go off process
3. local rearrangement to fill holes generated by particles moving off process
4. read in particles sent from other processors.

The system was receiving more "unexpected messages" than it could handle, exhausting internal MPI buffer space. The memory allocated for unexpected messages can be set via an environment variable, but this is a fragile solution in my experience. There is also a hard limit determined by the total amount of memory available to a process.

XGC-1: MPI problem fix

The performance for the original code was good (until it failed), so performance bar was set high.

		Seconds	
	processes	main_loop	shift (max, min, process 0)
Original:	8192	345	(50, 21, 26)
	16384	failed	-
Fix 1:	8192	411	(132, 96, 108)
	16384	483	(202, 162, 180)
Fix 2:	16384	459	(177, 134, 148)
Fix 4:	16384	465	(184, 141, 156)
Fix 5:	16384	468	(180, 142, 155)
Fix 7:	8192	397	(113, 77, 81)
	16384	458	(170, 131, 143)
FINAL:	8192	350	(59, 25, 27)
	16384	356	(63, 26, 35)

XGC-1: MPI problem fix

Final algorithm:

1. determine where to send particles (MPI_Alltoall OR MPI_Allreduce + point-to-point)
2. post all receive requests, optionally sending handshaking messages to sources
3. local rearrangement to fill holes generated by particles moving off process
4. send all particles that need to go off process, optionally waiting for handshaking message (flow control)
5. read in particles sent from other processors

Default (reported in results) is to use MPI_Alltoall and flow control. Further MPI optimizations are possible, but it is unclear how much more performance improvement is possible.

XGC-1: MPI overhead analysis

processes	main_loop	charge_comm	PETSc_solve	Shift
		(max, min) over processes		
256	345	(16, 6)	(18, 17)	(25, 20)
512	336	(28, 6)	(10, 9)	(26, 18)
1024	328	(33, 7)	(9, 8)	(28, 19)
2048	330	(48, 7)	(12, 11)	(38, 20)
4096	332	(69, 7)	(18, 16)	(50, 21)
8192	344	(82, 8)	(29, 26)	(56, 24)
16384	356	(95, 15)	(29, 27)	(63, 26)

These data (and others, not shown) indicate load imbalance in charge deposition routine and in computation leading into particle shift that increases with process count. PETSc solve is on fixed grid, so is solely MPI communication overhead as process count increases. Load imbalance is being addressed, but short term approach is to introduce OpenMP parallelism in order to decrease number of MPI processes.

OAK RIDGE National Laboratory

Office of Science U.S. DEPARTMENT OF ENERGY

XGC-1: OpenMP optimization

OpenMP is used effectively in GTC, another particle code used by the fusion simulation community, and the expectation was that XGC-1 would be similarly amenable to OpenMP parallelism. Reductions required splitting some loops, but introducing OpenMP was relatively straightforward. Performance was initially quite poor, with runtime increasing as a function of thread count for the worst case loops.

Empirical experiments (e.g., restructuring loops) on small core counts indicated that all performance issues arose from working with arrays that were accessed via a pointer in a structure. Changing these to allocatable arrays within the structure improved performance somewhat, but the problem was not eliminated until the arrays were added as arguments to the relevant subroutines (and not accessed via the structure within the routines). Fortunately, this involved only a few arrays and only a few routines were affected. This modification also improved performance when not using OpenMP.

OAK RIDGE National Laboratory

Office of Science U.S. DEPARTMENT OF ENERGY

XGC-1: Current Performance

processes	threads per process	Main_loop	charge_comm
8192	1	336	(81, 9)
2048	4	306	(48, 5)
16384	1	343	(81, 10)
4096	4	310	(66, 6)

where the same number of particles are assigned to each thread.

The OpenMP parallel implementation is not yet mature, but getting reasonable performance improvement. Next steps:

1. Load imbalance: For 16384 processes, 22% variation in number of particles. For 4096 processes, less than 2% variation. So particle allocation imbalance is only part of the computational load imbalance. Need to understand (and eliminate?) source of load imbalance.
2. OpenMP does not speed up I/O currently. Performance in diagnostic routines is partially a function of the number of particles per process (not thread), which is 4X larger when using OpenMP. ADIOS asynchronous I/O may make this unimportant.

Community Atmosphere Model

(Note: N == horizontal grid size, P == number of MPI processes.)

1. Fast reproducible global sum algorithm
   - Replaced “master computes” algorithm with distributed algorithm, reducing memory and computational complexity from O(N) to O(N/P). Reduced time from 4% of total execution time for large benchmark on 1664 XT processors to ~0.1% (50 times faster).
2. Determination of amount of memory needed to gather distributed data structure into master
   - Replaced O(PN) algorithm by O(N) algorithm. Reduced time from 20% of total execution time for large benchmark on 1024 BG/P processors to .03% (~1000 times faster).
3. Determination of whether distributed data structure has haloes
   - Replaced O(NlogN) algorithm executed each timestep with a single call during initialization. Reduced total execution time for large benchmark on 1024 BG/P processors by 3%.

All identified using profile data. Solutions based on complexity analyses and introduction of alternative algorithms.

Community Atmosphere Model

4. Identifying and eliminating unnecessary algorithmic restrictions on scalability:
   a. Limitations in one phase of the code need not limit another phase
   b. Identifying other work that can be computed simultaneous to a phase with limited parallelism (and modifying code to support this functional parallelism)
   - Issue: For certain problem instances, when more MPI processes are active in one phase of the code than in another, the cost of MPI communication between the phases increases *significantly* (on IBM BG/P, Cray XT4, and ATLAS infiniband cluster at LLNL). For example, on IBM BG/P: small problem (96x144x30) but with full atmospheric chemistry (108 tracers), pure MPI, 480 processes in "phase 1", process 0 data:

# of "phase 2" processes	480	1536	
Main_loop	2659 secs.	2933	(slower!)
phase 2 to phase 1	103 secs.	1586	(> 10X slower!)
phase 1 to phase 2	32 secs.	56	

OAK RIDGE National Laboratory

Office of Science U.S. DEPARTMENT OF ENERGY

CAM: MPI scaling problem

Different domain decompositions are used in the two phases, and the MPI communication between phases is essentially a transpose. The original algorithm was as follows:

1. Prepost all receive requests (using MPI_IRECV)
2. Send all data (using MPI_ISEND)
3. Wait for receive requests to complete
4. Wait for send request to complete

Upon closer examination of the code, we identified two possible sources for performance degradation:

- The ordering of the sends for all processes was to send to process 0, 1, 2, … thus maximizing the potential for contention.
- In one direction, some processes are only sending, not receiving, so the preposted receives may not be posted early enough to be effective.

Oak Ridge National Laboratory

Office of Science
U.S. Department of Energy

CAM: MPI scaling problem fix

To diagnose (and ultimately fix) the communication problem:

1. A number of runtime options were introduced:
   a. choice of MPI_SEND or MPI_ISEND
   b. flow control (sending handshaking messages after preposting receive requests; not sending data until receive handshake; using MPI_IRSEND or MPI_RSEND)
   c. limit on maximum number of nonblocking MPI requests
   d. Choice of MPI_ALLTOALLV or point-to-point implementation
2. a dimensional exchange ordering of the send/receive requests was introduced for the point-to-point implementation, to eliminate the hot spots.

Current default is point-to-point, MPI_SEND, flow control, with no limit on number of nonblocking MPI requests. This performances reasonably well across all scenarios that we have investigated.

CAM: MPI scaling problem fix

Results using “final solution” (same benchmark problem)

- BG/P: 256 MPI processes in phase 1, 1024 MPI processes in phase 2
  - original: 6.2 simulated years per day (SYPD)
  - new point-to-point: 9.2 SYPD
  - MPI_ALLTOALLV: 9.7 SYPD

  (Remember, only difference is in tranpose algorithm.)

  MPI_ALLTOALLV is not optimal on all platforms, however: (same benchmark problem, but utilizing same parallelism in both phases)
- IBM BGP, 480 MPI processes, 4 OpenMP threads per process
  - new point-to-point: 7.2 SYPD
  - MPI_ALLTOALLV: 7.5 SYPD
- Cray XT4, 480 MPI processes, 4 OpenMP threads per process
  - point-to-point: 21.6 SYPD
  - MPI_ALLTOALLV: 7.6 SYPD

CAM: XT4 Performance Variability

In CAM benchmarking, have documented as yet undiagnosed performance variability, but am assuming that it is due to interference from other users. This was an extreme case, but is happens often enough that I always try to “babysit” my benchmark runs.

- 256 MPI processes, 4 OpenMP threads per process
- Small problem (96x144x30), no atmospheric chemistry
- First run
  - 7.4 SYPD
  - phase 2 to phase 1 transpose: 82 seconds
- Second run (separate qsub later in the same day)
  - 64.4 SYPD
  - Phase 2 to phase 1 transpose: 5 seconds

How diagnose? Systems administrators were not able to identify anything unusual in their logs.

Oak Ridge National Laboratory

Office of Science U.S. Department of Energy

Conclusions

1. (Still) finding algorithmic limitations to scalability that are easily identified with relatively primitive tools when can run empirical experiments at an appropriate scale. Developing and evaluating alternative approaches often requires experimentation at scale.
2. Applications can easily consume local system resources as process count grows. Applications need to be aware of and control resource demands (e.g., MPI flow control).
3. “Defensive Programming” strategies continue to be important, and the importance of compile and runtime tuning options will likely increase as the optimal algorithms and implementations vary with platform, problem instance, core count, …
4. An experiment-centric approach to performance optimization appears to work even at scale, if ready access to the required resources is available. However, global system performance data can be important when interpreting individual application performance data.