Performance Modeling of HPC Applications

A. Snavely[a], X. Gao[a], C. Lee[a], L. Carrington[b], N. Wolter[b], J. Labarta[c], J. Gimenez[c], P. Jones[d]

[a]University of California, San Diego, CSE Dept. and SDSC, 9300 Gilman Drive, La Jolla, Ca. 92093-0505, USA

[b]University of California, San Diego, SDSC, 9300 Gilman Drive, La Jolla, Ca. 92093-0505, USA

[c]CEPBA, Jordi Girona 1-3, Modulo D-6, 08034 Barcelona, Spain

[d]Los Alamos National Laboratory, T-3 MS B216, Los Alamos, N.M. USA

This paper describes technical advances that have enabled a framework for performance modeling to become practical for analyzing the performance of HPC applications.

1. Introduction

Performance models of applications enable HPC system designers and centers to gain insight into the most optimal hardware for their applications, giving them valuable information into the components of hardware (for example processors or network) that for a certain investment of time/money will give the most benefit for the applications slated to run on the new system. The task of developing accurate performance models for scientific application on such complex systems can be difficult. In section 2 we briefly review a framework we developed [1] that provides an automated means for carrying out performance modeling investigations. In section 3 we describe ongoing work to lower the overhead required for obtaining application signatures and also how we increased the level-of-detail of our convolutions with resulting improvements in modeling accuracy. In section 4 we show how these technology advances have enabled performance studies to explain why performance of applications such as POP (Parallel Ocean Program) [2], NLOM (Navy Layered Ocean Model) [3], and Cobalt60 [4] vary on different machines and to quantify the performance effect of various components of the machines. In section 5 we generalize these results to show how these application's performance would likely improve if the underlying target machines were improved in various dimensions (as for example on future architectures).

2. Brief Review of a Convolution-based Framework for Performance Modeling

In a nutshell, we map a trace of program events to their expected performance established via separately measured or extrapolated machine profiles. We combine single-processor models along with the parallel communications model, to arrive at a performance model for a whole parallel application. We emphasize simplicity in the models (leaving out, for example, second order performance factors such as instruction level parallelism and network packet collisions) while applying these simple models at high resolution. A user of the framework can input the performance parameters of an arbitrary machine (either existing and profiled, or under-design and estimated) along with instruction/memory-access-pattern signatures and communications signatures for an application to derive a performance model. The convolver tools calculate the expected performance of the application on the machine in two steps, first by modeling the sections between communications events and then by combining these models into a parallel model that includes MPI communications. Detailed descriptions of our performance modeling framework can be found in papers online at the Performance Modeling and Characterization Laboratory webpages at `http://www.sdsc.edu/pmac/Papers/papers.html`

3. Reducing Tracing Time

For portability and performance reasons we ported our tracing tool, MetaSim Tracer, to be based on DyninstAPI [5] [6]. Previously it was based on the ATOM toolkit for Alpha processors. This meant applications could only be traced on Alpha-based systems. A more critical limitation was due to the fact that ATOM instruments binaries statically prior to execution. This means tracing cannot be turned on and off during execution. DyninstAPI is available on IBM Power3, IBM Power4, Sun, and Intel processors. It allows code instrumentation via runtime patching. The image of the running program can be modified during execution to add instrumentation. The instrumentation can be dynamically disabled. The opportunity was to enable a feature whereby MetaSim Tracer can sample performance counters by adding instrumentation during sample phases. The program can be de-instrumented between sample phases. Slowdown due just to minimal hooks left in the code to allow re-instrumentation should be greatly reduced between sample phases. An open question remained that we wished to answer before proceeding; whether application traces based on sampling could yield reasonably accurate performance models. Some previous work [7] showed this is possible and in recent work we also demonstrated it can via experiments with the ATOM based version of MetaSim Tracer. In these experiments we turned on and off the processing of instrumented sections (we could not actually turn off the instrumentation in the ATOM version, so we just switched off processing in the callback routines). In this way we were able to explore the verisimilitude of interpolated traces based on sampled data, and we showed that these could indeed be usefully accurate [8].

Encouraged by these results we then implemented a DyninstAPI version of MetaSim Tracer so that the duration and frequency of sampling periods (counted in CPU cycles) is under the control of the user. The user inputs two parameters: 1) SAMPLE = number of cycles to sample 2) INTERVAL = number of cycles to turn off sampling. The behavior of the program when sampling is turned off is estimated by interpolation. Thus MetaSim Tracer now enables a tradeoff between time spent tracing and verisimilitude of the resulting trace obtained via sampling. A regular code may require little sampling to establish its behavior. A code with very random and dynamic behaviors may be difficult to characterize even from high sampling rates. Practically, we have found techniques for generating approximated traces via sampling can reduce tracing time while preserving reasonable trace fidelity. Also we found that representing traces by a dynamic CFG decorated with instructions (especially memory instructions) characterized by memory access pattern can reduce the size of stored trace files by three orders of magnitude [9]. These improvements in the space and time required for tracing have now rendered full-application modeling tractable. In some cases it is possible to obtain reasonably accurate traces and resulting performance models from 10% or even 1% sampling.

4. Modeling Application Performance on HPC Machines

4.1. Parallel Ocean Program (POP)

The Parallel Ocean Program (POP) [2] was specifically developed to take advantage of high performance computer architectures. POP has been ported to a wide variety of computers for eddy-resolving simulations of the world oceans and for climate simulations as the ocean component of coupled climate models. POP has been run on many machines including IBM Power3, and IBM Power4 based systems, Compaq Alpha server ES45, and Cray X1. POP is an ocean circulation model that solves the three-dimensional primitive equations for fluid motions on the sphere under hydrostatic and Boussinesq approximations. Spatial derivatives are computed using finite-difference discretizations, formulated to handle any generalized orthogonal grid on a sphere, including dipole, and tripole grids that shift the North Pole singularity into land masses to avoid time step constraints due to grid convergence. The benchmark used in this study is designated 'x1' (not to be confused with the Cray X1 machine, one of the machines where we ran the benchmark); it has coarse resolution that is currently being used in coupled climate models. The horizontal resolution is one degree (320x384) and uses a displace-pole grid with the pole of the grid shifted into Greenland and enhanced resolution in the equatorial regions. The vertical coordinate uses 40 vertical levels with smaller grid spacing near the surface to better resolve the surface mixed layer. This configuration does not resolve eddies, and therefore it requires the use of computationally intensive subgrid parameterizations. This configuration is set up to be identical to the actual production configuration of the Community Climate System Model with the exception that the coupling to full atmosphere, ice and land models has been replaced by analytic surface forcing. We applied the modeling framework to POP. The benchmark does not run so long as to require sampling. Table 1 shows real vs. model-predicted wall-clock execution times for several machines at several processor counts. We only had access to a small 16 CPU Cray X1. The model is quite robust on all the machines modeled with an average error of only 6.75% where error is calculated as (Real Time - Predicted Time)/(Real Time) *100.

Table 1
Real versus Predicted-by-Model Wall-clock Times for POP Benchmark
Cray X1 at 16 processors had real time 9.21 seconds, predicted time 9.79 seconds, error 6.3 percent

# of CPUs	Real Time (sec)	Predicted Time (sec)	Error%
	Blue Horizon Power3 8-way SMP Colony switch		
16	204.92	214.29	-5%
32	115.23	118.25	-3%
64	62.64	63.03	1%
128	46.77	40.60	13%
	Lemeiux Alpha ES45 4-way SMP Quadrics switch		
16	125.35	125.75	0%
32	64.02	71.49	-11%
64	35.04	36.55	-4%
128	22.76	20.35	11%
	Longhorn Power4 32-way SMP Colony switch		
16	93.94	95.15	-1%
32	51.38	53.30	-4%
64	27.46	24.45	11%
128	19.65	15.99	16%
	Seaborg Power3 16-way SMP Colony switch		
16	204.3	200.07	2%
32	108.16	123.10	-14%
64	54.07	63.19	-17%
128	45.27	42.35	6%

4.2. Navy Layered Ocean Model (NLOM)

The Navy's hydrodynamic (iso-pycnal) non-linear primitive equation layered ocean circulation model [3] has been used at NOARL for more than 10 years for simulations of the ocean circulation in the Gulf of Mexico, Carribean, Pacific, Atlantic, and other seas and oceans. The model retains the free surface and uses semi-implicit time schemes that treat all gravity waves implicitly. It makes use of vertically integrated equations of motion, and their finite difference discretizations on a C-grid. NLOM consumes about 20% of all cycles on the supercomputers run by DoD's High Performance Computing Modernization Program (HPCMP) including Power3, Power4, and Alpha systems. In this study we used a synthetic benchmark called synNLOM that is representative of NLOM run with data from the Gulf of Mexico and has been used in evaluating vendors vying for DoD TI-02 procurements. Even though synLOM is termed a 'benchmark' it is really a representative production problem and runs for more than 1 hour un-instrumented on 28 CPUs (a typical configuration) on BH. Thus, in terms of runtime, it is an order-of-magnitude more challenging to trace than POP x1. We used 1% sampling and the resulting models yielded less than 5% error across all of the same machines

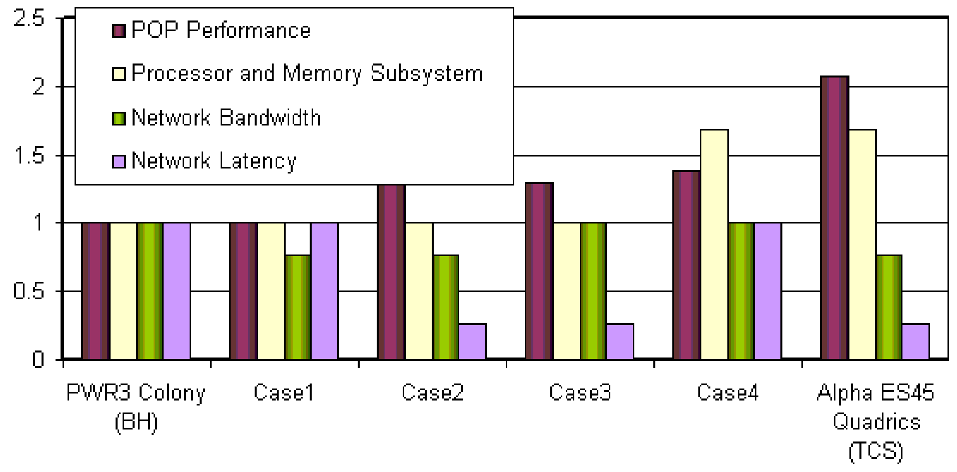

Figure 1. Modeled Contributions to Lemeuix.s (TSC) performance improvement over Blue Horizon on POP x1 at 16 CPUs.

as in the above POP study. We estimate a full trace would take more than a month to obtain on 28 processors! NLOM is reasonably regular and the low error percentages from 1% sampling do do not seem to justify doing a full trace although the code is important enough to DoD that they would provide a month of system time for the purpose.

4.3. Cobalt60

Cobalt60 [4] is an unstructured Euler/Navier-Stokes flow solver that is routinely used to provide quick, accurate aerodynamic solutions to complex CFD problems. Cobalt60 handles arbitrary cell types as well as hybrid grids that give the user added flexibility in their design environment. It is a robust HPC application that solves the compressible Navier-Stokes equations using an unstructured Navier-Stokes solver. It uses Detached-Eddy Simulation (DES) which is a combination of Reynolds-averaged Navier-Stokes(RANS) models and Large Eddy Simulation (LES). We ran 7 iterations of a tunnel model of an aircraft wing with a flap and endplates with 2,976,066 cells that runs for about an hour on 4 CPUs of BH. We used a 2-step trace method to ascertain in the first phase that 70% of the time is spent in just one basic block. We then applied 1% sampling to this basic block and 10% sampling to all the others in the second step of MetaSim tracing. We verified models for 4, 32, 64, and 128 CPUS on all the machines used in the previous study with average of less than 5% error.

5. Performance Sensitivity Studies

Reporting the accuracy of performance models in terms of model-predicted time vs. observed time (as in the previous section) is mostly just a validating step for obtaining confidence in the model. More interesting and useful is to explain and quantify performance differences and to play 'what if' using the model. For example, it is clear from Table 1 above that Lemeuix is faster across-the-board on POP x1 than is Blue Horizon. The question is why? Lemeuix has faster processors (1GHz vs. 375 MHz), and a lower-latency network (measured ping-pong latency of about 5 ms vs. about 19 ms) but Blue Horizon.s network has the higher bandwidth (ping-pong bandwidth measured at about 350 MB/s vs. 269 MB/s with the PMaC probes). Without a model one is left with a conjecture 'I guess POP performance is more sensitive to processor performance and network latency than network bandwidth'.

With a model that can accurately predict application performance based on properties of the code and the machine, we can carry out precise modeling experiments such as that represented in Figure 1. We model perturbing the Power3-based, Colony switch Blue Horizon (BH) system into the Alpha ES640-based, Quadrics switch system (TCS) by replacing components. Figure 1 represents a series

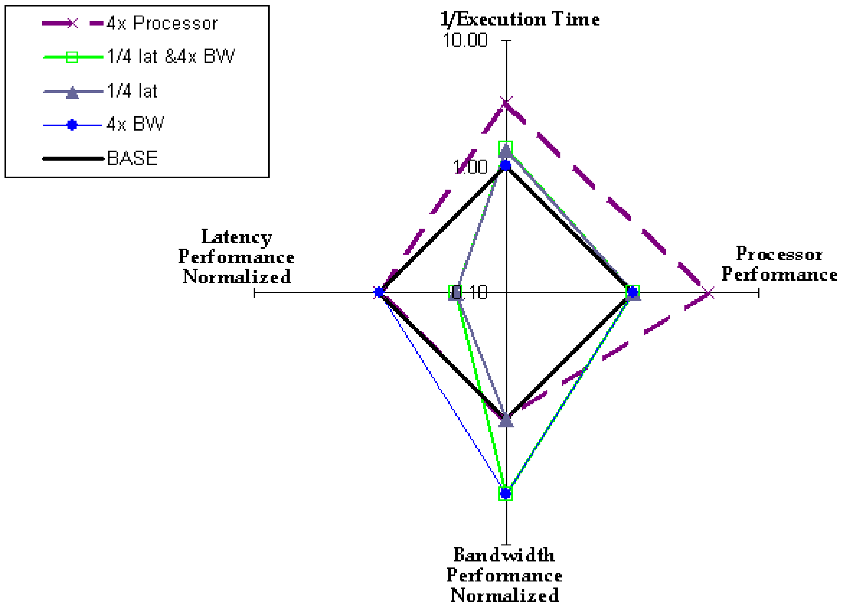

Figure 2. POP Performance Sensitivity for 128 cpu POP x1.

of cases modeling the perturbing from BH to TCS, going from left to right. The four bars for each case represent the performance of POP x1 on 16 processors, the processor and memory subsystem performance, the network bandwidth, and the network latency all normalized to that of BH. In Case 1, we model the effect of reducing the bandwidth of BH's network to that of a single rail of the Quadrics switch. There is no observable performance effect as the POP x1 problem at this size is not sensitive to a change in peak network bandwidth from 350MB/s to 269MB/s. In Case 2 we model the effect of replacing the Colony switch with the Quadrics switch. There is a significant performance improvement due to the 5 ms latency of the Quadrics switch versus the 20 ms latency of the Colony switch. This is because the barotropic calculations in POP x1 at this size are latency sensitive. In Case 3 we use Quadrics latency but Colony bandwidth just for completeness. In Case 4 we model keeping the Colony switch latencies and bandwidths but replacing the Power3 processors and local memory subsystem with Alpha ES640 processors and their memory subsystem. There is a substantial improvement in performance due mainly to the faster memory subsystem of the Alpha. The Alpha can load stride-1 data from its L2 cache at about twice the rate of the Power3 and this benefits POP x1 a lot. The last set of bars show the values of TCS performance, processor and memory subsystem speed, network bandwidth and latency, as a ratio to BH's values. The higher-level point from the above exercise is that the model can quantify the performance impact of each machine hardware component. We carried out similar exercise for several sizes of POP problem and for NLOM, Cobalt60, and could do so for any application modeled via the framework. Larger CPU count POP x1 problems become more network latency sensitive and remain not-very bandwidth sensitive.

As an abstraction from a specific architecture comparison study such as the above, we can use the model to generate a machine-independent performance sensitivity study. As an example, Figure 2 indicates the performance impact on a 128 CPU POP x1 run for quadrupling the speed of the CPU and memory subsystem (lumped together we call this processor), quadrupling network bandwidth, cutting network latency by 4, and various combinations of these four-fold hardware improvements. The axis are plotted logscale and normalized to 1, thus the solid black quadrilateral represents the execution time, network bandwidth, network latency, CPU and memory subsystem speed of BH. At this size POP x1 is quite sensitive to processor (faster processor and memory subsystem), somewhat sensitive to latency because of the communications-bound with small-messages barotropic portion of the calculation and fairly insensitive to bandwidth. With similar analysis we can 'zoom in' on the processor performance factor. In the above results for POP, the processor axis shows modeled execution time decreases from having a four-times faster CPU with respect to MHz (implying 4X floating-point issue rate) but also implicit in '4X node' is quadruple bandwidth and 1/4th latency to all levels of the memory hierarchy (unfortunately this may be hard or expensive to achieve architecturally!). We explored how much faster a processor would perform relative to the Power3 processor for synNLOM if it had 1) 2X issue rate 2) 4X issue rate, 3) 2X issue rate and 2X faster L2 cache 4) base issue rate of 4*375 MHz but 4X faster L2 cache. Space will not allow the figure here but qualitatively we found synLOM at this size is compute-bound between communication events and would benefit a lot just from a faster processor clock even without improving L2 cache. Not shown but discoverable via the model is that synNLOM is somewhat more network bandwidth sensitive than POP because it sends less frequent, larger messages. With similar analysis we found Cobalt60 is most sensitive to improvements in the processor performance at this input size and this remains true at larger processor counts. The higher-level point is that performance models enable 'what-if' examinations of the implications of improving the target machine in various dimensions.

6. Conclusion

A systematic method for generating performance models of HPC applications has advanced via efforts of this team and has begun to make performance modeling systematic, time-tractable, and thus generally useful for performance investigations. It is reasonable now to make procurement decisions based on the computational demands of the target workload. Members of this team are now working closely with the Department of Defense High Performance Computing Modernization Program Benchmarking Team to effect TI-05 procurements by just such criteria. It is reasonable now to tune current systems and influence the implementation of near-future systems informed by the computational demands of the target workload; team members are collaborating in the DARPA HPCS program to influence the design of future machines.

7. Acknowledgements

This work was sponsored in part by the Department of Energy Office of Science through SciDAC award High-End Computer System Performance: Science and Engineering. This work was sponsored in part by the Department of Defense High Performance Computing Modernization Program office through award HPC Benchmarking.

REFERENCES

[1] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia A. Purkayastha, A Framework for Performance Modeling and Prediction, SC2002.

[2] See `http://www.acl.lanl.gov/climate/models/pop/current_release/UsersGuide.pdf`

[3] A. J. Wallcraft, The Navy Layered Ocean Model Users Guide, NOARL Report 35, Naval Research Laboratory, Stennis Space Center, MS, 21 pp, 1991.

[4] See `http://www.cobaltcfd.com/`

[5] See www.dyninst.org

[6] J. K. Hollingsworth, An API for Runtime Code Patching, IJHPCA, 1994.

[7] J. L., Hennessy, D. Ofelt, .Efficient Performance Prediction For Modern Microprocessors., ACM SIGMETRICS Performance Evaluation Review, Volume 28, Issue 1, June 2000.

[8] L. Carrington, A. Snavely, N. Wolter, X. Gao, A Performance Prediction Framework for Scientific Applications, Workshop on Performance Modeling and Analysis - ICCS, Melbourne, June 2003.

[9] X. Gao, A. Snavely, Exploiting Stability to Reduce Time-Space Cost for Memory Tracing, Workshop on Performance Modeling and Analysis - ICCS, Melbourne, June 2003.