A Case Study Using Automatic Performance Tuning for Large-Scale Scientific Programs

I-Hsin Chung
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
ihchung@us.ibm.com

Jeffrey K. Hollingsworth
Department of Computer Science
University of Maryland
College Park, MD 20742
hollings@cs.umd.edu

***Abstract*— Active Harmony is an automated runtime performance tuning system. In this paper we describe several case studies of using Active Harmony to improve the performance for scientific libraries and applications. We improved the tuning mechanism so it can work iteratively with benchmarking runs. By tuning the computation and data distribution, Active Harmony helps applications that utilize the *PETSc* library to achieve better load balance and to reduce the execution time up to 18%. For the climate simulation application *POP* using 480 processors, the tuning results show that by changing the block size and parameter values, the execution time is reduced up to 16.7%. Active Harmony is able to improve *GS2*, a plasma physics code, up to a factor of 5.1 times faster. The experiment results show that the Active Harmony system is a feasible and useful tool to automated performance tuning for scientific libraries and applications.**

I. Introduction

Scientific applications today use different platforms from clusters, to SMP's to vector machines. In order to fully utilize these systems, performance tuning is important.

Currently, performance tuning for scientific applications heavily relies on a small group of experienced people. A commonly used mechanism for scientific program performance tuning is to work on "representative short runs" or benchmarking runs. In other words, the scientific program is run with meaningful input/configuration for a short period of time. Based on the performance observed, the experienced people may make some changes and then run the program again. This process repeats until the performance is acceptable. After that, the program is tuned and ready for production runs. The "representative" or meaningful input/configuration is usually decided by the scientists who developed the scientific program or who have in-depth domain knowledge. This tuning process is usually time consuming and not cost-effective. In addition, due to the fact that frequently these applications have large computational demands and thus need to be run on large-scale parallel computers, even a small percentage improvement in the execution time will reduce the cost dramatically. Alternatively, with improved execution time, the program can also achieve better results such as higher resolution, better precision or use a larger data set.

An important method that is commonly used in performance tuning is to adjust the configurations to achieve better performance. For example, a configuration parameter can be data distribution. If the data layout is properly aligned, the communication cost can be minimized such that overall execution time is reduced. Likewise, performance can also be improved by better load balancing.

Active Harmony is designed for runtime performance tuning in a dynamic environment. To do this, domain knowledge may be needed in order to decide what parameters can be tuned and their valid ranges. Active Harmony allows general users to have the benefits of performance tuning without in-depth domain knowledge. With Active Harmony, application developers can include tuning into the design and specify tunable parameters. Therefore the general users can adapt the application easily for different environments. In this paper, we apply the Active Harmony system to some widely used scientific libraries and applications. The search space for the tuning is huge and thus cannot be done manually. We show how Active Harmony helps to improve performance for scientific applications. By changing

the configuration, we can reduce the execution time of scientific libraries and applications significantly.

This paper differs from our previous work [11], [13] in that we apply Active Harmony to large-scale scientific applications running on high performance computers. The tuning experiments are done with computing/data distributions and parameter configurations. We extended the tuning mechanism from runtime "on-line" to "off-line" using representative short runs[1]. We show that general users may easily apply Active Harmony to scientific applications without in depth domain knowledge.

Active Harmony suggests to use on-line tuning if the parameter can and needs be changed during runtime. Off-line tuning applies to parameters that are fixed and cannot be changed during runtime (e.g., parameters that are read once when the program starts). This also applies for parameters that can be changed during runtime but are insensitive to workload (e.g., some parameters need to be changed once per machine as part of porting efforts).

The structure of this paper is organized as follows: Section II gives an brief overview of the Active Harmony tuning system. Section III discusses issues involved when applying the Active Harmony. Section IV, V, and VI demonstrate how Active Harmony is applied to three different scientific library and applications (*PETSc*, *POP*, and *GS2*) respectively to improve the performance. Section VII discusses issues involved when applying the Active Harmony. Related work is given in Section VIII and Section IX concludes the paper.

II. Active Harmony

To provide automatic performance tuning, we developed the Active Harmony system [11]–[13], [19], [20]. Active Harmony is an infrastructure that allows applications to become tunable by applying minimal changes to the application and library source code. This adaptability provides applications with a way to improve performance during a single execution based on the observed performance. The types of things that can be tuned at runtime range from parameters such as the size of a read-ahead buffer to what algorithm is being used (e.g., heap sort vs. quick sort).

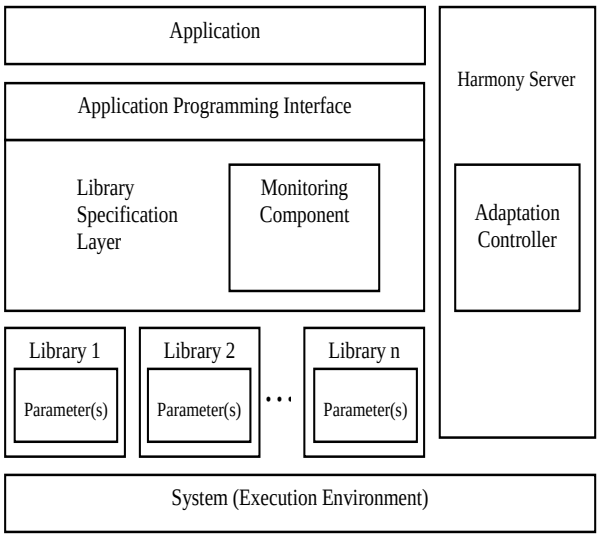


Fig. 1. Active Harmony Automated Tuning System

Figure 1 shows the Active Harmony automated runtime tuning system. The Library Specification Layer provides a uniform API to library users by integrating different libraries with the same or similar functionality.

The Adaptation Controller is the main part of the Harmony server. This component manages the values of the different tunable parameters provided by the applications and changes them for better performance. The kernel of the adaptation controller is a tuning algorithm. The algorithm is based on the Nelder-Mead simplex method [23] for a finding a function's minimum value.

In the Active Harmony system, we treat each tunable parameter as a variable in an independent dimension[2]. Each configuration which is a set of tunable parameters is represented as a point in the search space. The objective function can be decided by the user. The algorithm makes use of a simplex, which is a geometrical figure defined by $k + 1$ connected points in a k-dimensional space. In 2-D space, the simplex is a triangle, and for the 3-D space the simplex is a non-degenerated tetrahedron. The simplex method approximates the extreme of

[1]"On-line" tuning refers to program configuration changes during runtime without stopping/restarting. "Off-line" with representative short runs refers to the program running for a short period of time; stopping; changing configuration and then restarting.

[2]For dependent variables, we use techniques developed in [12] to make proper parameter value selection for tuning.

a function by considering the worst point of the simplex and forming its symmetrical image through the center of the opposite (hyper) face. At each step a better point replaces the worst points and thus moves the simplex towards the extreme, in our case towards the minimum.

The original simplex algorithm assumes a well-defined function and works in a continuous space. However, neither of these assumptions holds in our situation. Thus we have adapted the algorithm by simply using the resulting values from the nearest integer point in the space to approximate the performance at the selected point in the continuous space.

Due to the large search domain for possible configurations, the major challenge when users apply Active Harmony to a large-scale system is the time it takes for tuning. A lengthy tuning process will make the tuning results unusable since the system or the environment may have changed by the time the algorithm finishes. In this paper, we apply the Active Harmony to scientific library and applications on large-scale high performance computers. It helps the application user to utilize the systems in a reasonable amount of time. In addition, it encourages developers to make libraries and applications tunable to adapt to different platforms and environments.

III. Performance Tuning

One contribution of this paper is that we added off-line tuning to Active Harmony so it can tune scientific libraries and applications iteratively.

In particular, we added the ability to use multiple representative short runs (e.g., benchmarking runs) and make tuning modifications between runs. This approach complements our existing technique of adding calls to the Active Harmony API to the application code to allow parameter changes during execution. Using representative short runs requires minimum or no modifications to the scientific library or the application. For some applications, there are parameters that are only referenced once after the program starts execution (e.g., parameters read from the configuration script file). Rather than make more extensive changes to the application, the Active Harmony system uses one representative short run as one tuning iteration.

Our experiments take all costs of parameter changes (including applications needed to be re-run and their warm up time) into consideration.

This "off-line" and "iterative" tuning mechanism is also useful if the parameter is hard-coded in the application. If the parameter is hard-coded into the binary program and there is no access to application source code, Active Harmony helps with performance tuning by adjusting configurations (e.g., inputs and topologies).

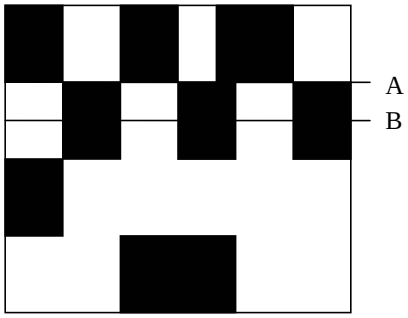
Active Harmony also provides an API for developers to make libraries and applications tunable (i.e., "on-line" and "iterative" tuning mechanism). For long-running applications where representative short runs are not available, the developers can easily hook up the application with Active Harmony tuning server. During the runtime, Active Harmony tuning server adjusts the parameter values to achieve better performance.

IV. *PETSc* Library

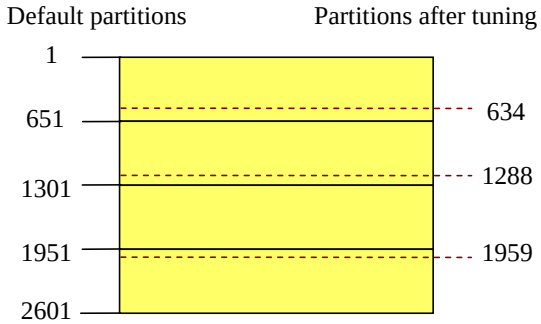
PETSc [4]–[6] (Portable, Extensible Toolkit for Scientific Computation) is a suite of data structures and routines for the scalable (parallel) solution of scientific application modeled by partial differential equations. *PETSc* is intended for use in large-scale application projects. Software packages that use or interface to *PETSc* include TAO [7], SCIRun [28], Magpar [27], libMesh [1], and Snark [2], [3]. It is widely used in optimization, biology, computational fluid dynamics, and wave propagation.

PETSc uses MPI for message communication. It integrates architecture dependent optimized libraries such as BLAS and LAPACK. It includes parallel linear and nonlinear equation solvers that can be easily integrated into C, C++, and Fortran programs. *PETSc* also provides interfaces to Matlab and Mathematica. From a performance point of view, it allows users to have detailed control over the solution process. For example, the user may specify the details of matrix decomposition for data storage or array distribution for computation. This makes performance tuning for this library interesting and challenging since those details are environment and problem dependent.

We applied the Active Harmony to two *PETSc* examples to show its tuning ability. For each example, it requires about 10 lines of modifications in the program to make it tunable via our tuning server. The first example solves a linear system in



(a)

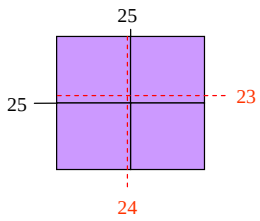


(b)

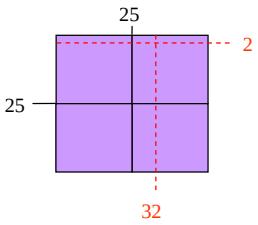
Fig. 2. Matrix decomposition

parallel with SLES (linear equation solver). The key point we are interested in is the matrix decomposition. Since decomposition affects data locality, it changes the amount of communication throughout the computation and thus has a dramatic impact on the performance. The concept is shown in Figure 2(a), the black blocks represent non-zero elements of the matrix. Data locality is improved if the matrix decomposition boundary uses the line A rather than using line B (i.e., dense sub-matrices are not spread across multiple nodes).

We made slight modifications to the source code (so the boundary is read from a configuration file instead of hard-coded) to allow Active Harmony to change the boundary for matrix decomposition. The total number of partitions is pre-defined (4 in this example). Each partition has at least one row and the number of rows for a single partition can be as small as one. Figure 2(b) shows the results for a small sample program running on four processing nodes. The default configuration (solid lines) decomposes the matrix into four even size partitions. After tuning, the result (dashed lines) shows that by changing the decomposition boundaries, the



(a) Homogeneous nodes



(b) Heterogeneous nodes

Fig. 3. Computation distribution

performance is improved. We also ran with a matrix size of $21,025 \times 21,025$ using 32 processing nodes. This results in an 18% improvement in execution time after tuning.

We also applied the Active Harmony to a problem with larger size. We use a matrix of size $90,601 \times 90,601$ (also with 32 processing nodes) as the input to the program. The full search space has $O(10^{100})$ points. With techniques we developed in our previous work [12], Active Harmony takes 120 iterations (tuning steps) to achieve a similar improvement (in the execution time) as the smaller matrix size problem (i.e., 15 – 20%).

The second *PETSc* example is a nonlinear driven cavity with multiple grids in two dimensions. The 2-D driven cavity problem is solved using a velocity-vorticity formulation. It uses the SNES (non-linear equation solver) object in the *PETSc* library. The Harmony tuning involves computation distribution. The problem consists of numerous grid points. The tunable parameter is how the grid points are distributed among processing nodes. The default configuration divides the grid points into distributed arrays with equal size. This works well in general when the processing nodes are homogeneous (i.e., all the processing nodes have the same processor type and speed). When using heterogeneous processing nodes (where there are nodes with different characteristics), the performance will be influenced dramatically by the layout of the computing grid points.

Figure 3 shows the configurations for a small example problem before and after tuning when using homogeneous and heterogeneous processing

nodes[3]. This problem consists of 2,500 grid points with 4 processing nodes. The solid lines are the default configurations and the dashed lines are the results after tuning. The distribution is different for homogeneous and heterogeneous nodes. By comparing Figure 3(a) and 3(b), it can be seen that when the processing nodes are homogeneous, the grid points should be divided into distributed arrays with equal size and in the heterogeneous environment, the system should try to utilize the processing nodes that have more computational powerful (the bottom two nodes).

We applied Active Harmony to the computation distribution example with 40,000 grid points using 32 processors (where the search space has $O(10^{36})$ points). As a result of tuning, up to an 11.5% improvement in the execution time (compared to default partitioning without tuning) was observed.

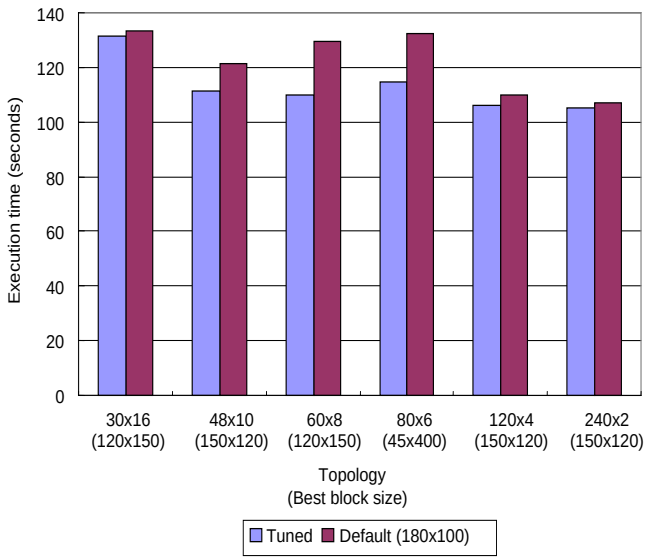


Fig. 4. Block size tuning of *POP* running on 480 processors using different number of CPUs per node.

V. Parallel Ocean Program (*POP*)

The Parallel Ocean Program (*POP*) [16], [17], [29] was developed at Los Alamos National Laboratory, and is a descendant of the Bryan-Cox-Semtner class of ocean models first developed at the NOAA (National Oceanic and Atmospheric Administration) Geophysical Fluid Dynamics Laboratory in Princeton, NJ in the late 1960s [10]. *POP* is currently used by the Community Climate System Model (CCSM) as the ocean component. *POP* solves three-dimensional primitive equations for fluid motions on a sphere using hydrostatic and Boussinesq approximations. Spatial derivatives are computed using finite-difference discretizations which are formulated to handle any generalized orthogonal grid on a sphere, including dipole and tripole grids.

We improved the performance (execution time) by enabling Active Harmony to adjust the block (a group of grid points) size and parameter values. The problem size is $3,600 \times 2,400$ grid points. The program divides the problem into 480 blocks (processors) and runs on a large SP-3 at NERSC (National Energy Research Scientific Computing Center). The nodes of the NERSC system consist of 16-way SMP's. One of the tuning criteria is how many processors per node to use. The Processors per node topology will affect the data locality and communication overhead. With better block size, data locality is improved and communication overhead is reduced. Therefore the overall performance (execution time) is improved. The default configuration came with 180×100 as the size for each block.

Figure 4 shows the tuning result. There are two set of labels in the x-axis. The first set of labels represents the topology for processing nodes and the second set of labels in the parenthesis represents the best block size found. The two different bars represent the performance for layouts before and after tuning. The first bar is the performance for the layout found (block size within the parenthesis on the x-axis) after tuning. The second bar is the performance using the default layout (180×100). Consider the second set of bars here. In this application, 48×10 indicates that the topology used in the experiment is 48 nodes with 10 processors on each node and the best block size found by tuning is 150×120. The figure shows that there is no single block size good for all topologies and the block size should be adjusted accordingly (block size 120×150 is best for topologies 30×16, 60×8; block size 150×120 is best for topologies 48×10, 120×4, and 240×2; block size 45×400 is

[3]In Figure 3(a), four nodes are identical Intel®Pentium™4 processors where in Figure 3(b), bottom two nodes are identical Intel®Pentium™4 processors and top two nodes are identical Intel®Pentium™II processors.

TABLE I

PARAMETER CHANGES THROUGH ITERATIONS[4]

Iteration	Parameter	Change from	To
0	(use default configuration)		
1	*num_iotasks*	1	32
2	*hmix_momentum_choice*	*anis*	*del2*
3	*hmix_tracer_choice*	*gent*	*del2*
4	*kappa_choice*	*constant*	*variable*
5	*slope_control_choice*	*notanh*	*clip*
6	*hmix_alignment_choice*	*east*	*grid*
7	*state_choice*	*jmcd*	*linear*
8	*state_range_opt*	*ignore*	*enforce*
9	*ws_interp_type*	*nearest*	*4point*
10	*shf_interp_type*	*nearest*	*4point*
11	*sfwf_interp_type*	*nearest*	*4point*
12	*ap_interp_type*	*nearest*	*4point*

TABLE II

PARAMETER TUNING

Parameter	Default	After tuning
num_iotasks	1	4
hmix_momentum_choice	*anis*	*del2*
hmix_tracer_choice	*gent*	*del2*
kappa_choice	*constant*	*variable*
slope_control_choice	*notanh*	*clip*
hmix_alignment_choice	*east*	*grid*
state_choice	*jmcd*	*linear*
state_range_opt	*ignore*	*enforce*
ws_interp_type	*nearest*	*4point*
shf_interp_type	*nearest*	*4point*
sfwf_interp_type	*nearest*	*4point*
ap_interp_type	*nearest*	*4point*

best for topology 80×6). With tuning only for the block size, the execution time can be reduced up to 15%. This shows that as the processors topology changes, the layout needs to be changed for better performance. Similarly, the same scientific program often runs on different platforms with a different number of processors per node, the results in this experiment show that the program should be tuned based on the machine configuration to achieve better performance.

Besides changing the block size, we further apply the Active Harmony system to adjust the parameter values using 32 processors (8 nodes, 4 processors/node) on Hockney at NERSC. The *POP* program has numerous parameters and there are about 20 parameters that are performance related. There are 2 to 4 possible values for each of the parameters. This makes the search space fairly large. However, the tuning results show that the Active Harmony system can achieve a 12.1% improvement in execution time after trying just 12 configurations. In addition, the best improvement is 16.7% improvement in execution time after 27 iterations. Table I shows how the parameter values have changed for each of the initial 12 iterations (only parameters whose value change are listed for each iteration). Table II shows the values for parameters that are changed before and after tuning (before and after 27 iterations). In this application a harmony iteration is one simulation run. Some of these parameters may affect scientific results. Ultimately tuning scientific programs requires assistance from experts with domain knowledge to make the tuning results useful and practical. We include these parameters in this experiment since our primary goal was to study the scalability of the harmony system, and not the output of the program being tuned.

VI. *GS2*

GS2 [15], [21] is a physics application, developed to study low-frequency turbulence in magnetized plasma. It is typically used to assess the microstability of plasmas produced in the laboratory and to calculate key properties of the turbulence which results from instabilities. It is also used to simulate turbulence in plasmas which occur in nature, such as in astrophysical and magnetospheric systems. Each of these modes uses the same simulation code on radically different time and space scales. The simulation involves billions of mesh points. We tuned the program with two different collision modes controlled by the *collision_model* variable (that controls which collision operator is used).

Our primary tuning parameter was "data layout". The program already had the ability to make a runtime selection of how to layout the data. The primary developer of the code had done some experiments to select a reasonable default layout. However, without an automated tool like Active Harmony, he was unable to fully tune the layout.

The initial analysis was done using 128 processors on NERSC Seaborg (8 nodes, 16 processors/node) system. In order to reduce the execution time, we applied Active Harmony to tune the

[4]Each row shows only the parameter that changes; all the rest of parameters remain the same compared to the previous iteration.

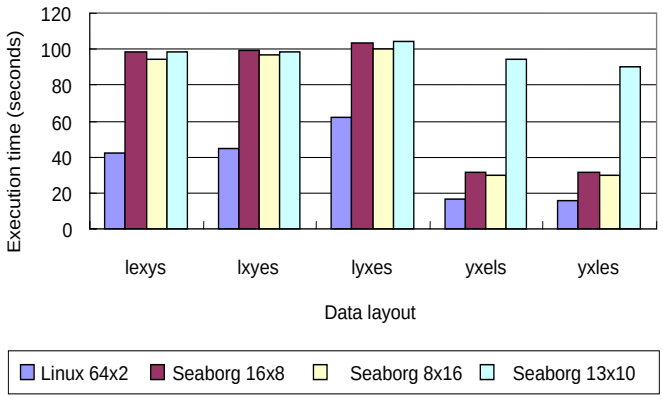


Fig. 5. *GS2* layout tuning with different environment[5]

program. By changing the data layout, the program execution time was reduced from 55.06s to 16.25s ($3.4\times$ faster) without collision mode and from 71.08s to 31.55s ($2.3\times$ faster) with collision mode. This is for a typical benchmarking run of 10 time steps. Production runs tend to have 1,000 or more time steps, and would see a similar improvement. Based on our results, the GS2 team has changed their default data layout to the ones recommended by Active Harmony.

In Figure 5, we compare the tuning results with different topologies on Seaborg and a result from a Linux cluster. The Linux cluster has 64 nodes; each node is equipped with dual Intel®Xeon™2.66GHz processors (with Hyper Threading enabled), 2GBytes main memory and a Myrinet network. In this experiment we consider different data layouts. The data layout is specified with five variables x,y,l,e, and s. The variables x and y are the spatial coordinates; l and e are velocity coordinates and s is the particle specie. The notation indicates the order of the dimensions of the primary 5-dimensional array in the simulation. The default data layout used by *GS2* is $lxyes$. In the figure, it shows that when the data can be aligned properly with the topology (Linux 64×2, Seaborg 16×8, Seaborg 8×16), using the right data layout ($yxles$, $yxels$) will improve the performance significantly.

We then proceeded to further improve the performance (execution time) on the Linux cluster. Based on the layout tuning result, we used Active Harmony

[5]The label $A\times B$ follows the machine name represents A nodes and B processors per node.

TABLE III

GS2 TUNING RESULT FOR BENCHMARKING RUN

Benchmarking run with "*lxyes*" layout		
Tuning method (*negrid*,*ntheta*,*nodes*)	Tuning time (iterations)	Tuning result - seconds (improvement %)
Default - no tuning (16,26,32)	–	43.7
Tuned version (8,22,8)	8	18.4 (57.9%)
Benchmarking run with "*yxles*" layout		
Tuning method (*negrid*,*ntheta*,*nodes*)	Tuning time (iterations)	Tuning result - seconds (improvement %)
Default - no tuning (16,26,32)	–	16.4
Tuned version (8,22,8)	9	14.8 (9.8%)

TABLE IV

GS2 TUNING RESULT FOR PRODUCTION RUN

Production run with "*lxyes*" layout		
Tuning method (*negrid*,*ntheta*,*nodes*)	Tuning time (iterations)	Tuning result - seconds (improvement %)
Default - no tuning (16,26,32)	–	1480.3
Tuned version (10,20,28)	9	244.2 (83.5%)
Production run with "*yxles*" layout		
Tuning method (*negrid*,*ntheta*,*nodes*)	Tuning time (iterations)	Tuning result - seconds (improvement %)
Default - no tuning (16,26,32)	–	384.9
Tuned version (10,16,18)	11	240.8 (37.4%)

to tune the program first using benchmarking runs (10 time steps) and then with input for production runs (1,000 time steps). There are three tunable parameters: $ntheta$ (number of grid points per 2π segment of field line), $negrid$ (energy grid), and the $nodes$ (number of nodes). These parameters were identified by the application developer who is the expert with domain knowledge. The tuning result for the benchmarking runs is summarized in Table III and for production runs in Table IV. In the experiments, we compare the tuning time (iterations) and results with and without tuning. The three values in the first column are the parameter values after tuning. There is larger improvement when the data layout is $lxyes$ compared to a better layout $yxles$ (that we find in the first part of the experiment). However, starting with the better data layout, Active Harmony achieves a better overall

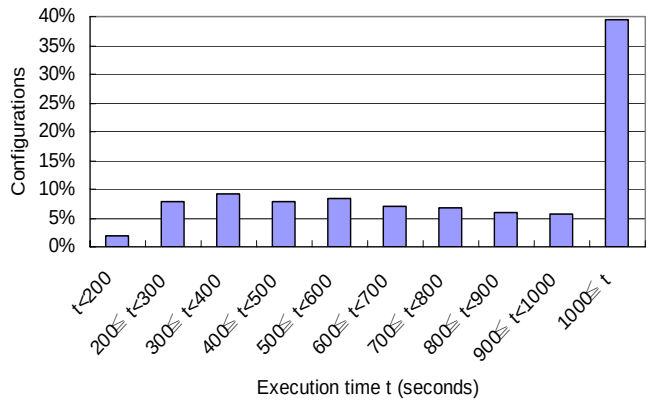


Fig. 6. Performance distribution for *GS2* configurations

performance result.

The experimental results also suggest that proper data alignment with the number of processors is the major factor deciding the performance. The order of the dimensions should be adjusted so the data can be aligned with the number of nodes and thus reduce the communication overhead. Therefore even with poor data layout such as $lxyes$, by adjusting the resolution and number or processors, it can still achieve comparable performance results.

While changing $negrid$ and $ntheta$ may affect the simulation resolution, the dramatic performance gains possible warrant considering using such parameters. As directed by the program developer, all the parameter value ranges used for tuning in this experiment will generate acceptable simulation resolutions. Practical scientific program tuning ultimately involves experts with domain knowledge who can make informed choices about these tradeoffs. If these tradeoffs can be quantified, other metrics such as fidelity and scheduling policy can also be specified and integrated into the objective function so the system can automate this tradeoff.

In order to compare the tuning result with the search space ($O(10^5)$ possible configurations) and to understand how well Active Harmony does the tuning, we also explore the whole search space using systematic sampling (i.e., using configurations that are evenly distributed in the whole search space) for the production runs. We explore ($O(10^4)$ configurations and the performance distribution is shown in Figure 6.

The best configuration found in this systematic sampling is $(negrid, ntheta, nodes) = (8, 16, 32)$ and its performance is 125.8s. However, these are rare points and there are only few configurations (less than 2%) in the whole search space with an execution time less than 200 seconds.

Compared to the performance gathered from systematic sampling, the configuration found by Active Harmony is within the top 5% of the configurations. Tuning using Active Harmony helps the *GS2* run $3.4\times$ faster. A lesson learned is that there are still a few configurations with better performance. In the future, we hope to develop techniques to find these configurations with less exhaustive techniques. Finding those configurations can be time-consuming and thus not cost-effective. By investing a small amount of effort, Active Harmony helps to tune the *GS2* program to produce significantly better performance.

In conclusion we applied two different techniques to tuning *GS2*: data distribution and parameters manipulation. Taken together these two techniques reduced the runtime of *GS2* by a factor of 5.1.

VII. Discussion

Active Harmony helps scientific libraries and applications adapt on different platforms and/or with different workloads. It shows the potential of tuning a large-scale system automatically. The tuning includes parameter adjustment inside the programs and explicit configuration changes to improve performance (e.g., better load balancing, less communication). As shown in the experimental results, the performance improvement is significant. This performance improvement is difficult to achieve by tuning each single component independently since it is extremely difficult to decide the contribution of each individual component to the performance of the whole application. Another advantage is that the user does not need to have detailed insight knowledge about each component when adapting the application to a new runtime environment. The user can simple apply the Active Harmony system to all the parameters that may be performance related.

Tuning for scientific programs is a challenging task due to numerous parameters with many possible values. Since the search space is too large, the optimal configuration may not be available in a

reasonable tuning time period. Even using sampling to understand the whole search space the experiment described in Section VI can take months of CPU time exploration for real scientific applications. Active Harmony searches for a good configuration intelligently to reduce the tuning time.

The selection between on-line and off-line tuning depends on the characteristics of the parameters. If a parameter can be changed during the runtime and is sensitive to the workload, on-line tuning is proposed. On the other hand, if a parameter cannot be changed during runtime (e.g., size of static memory allocation) or is insensitive to the workload, it might be fine to tune it once per run or once per machine (i.e., as part of the tuning effort).

The tradeoff between accuracy and performance improvement is an important issue in performance tuning. For scientific applications, there are some configurations that help to improve the performance but the output generated can be less accurate. This tradeoff ultimately involves experts with domain knowledge who can make informed choices. If these tradeoffs can be quantified, other metrics such as fidelity and scheduling policy can also be specified and integrated into the objective function so the system can automate this tradeoff.

VIII. Related Work

Autopilot [25], [26] allows applications to be adapted in an automated way. Autopilot (developed at the University of Illinois, Urbana-Champaign) integrates dynamic performance instrumentation and on-the-fly performance data reduction with configurable, malleable resource management algorithms. It has a real-time adaptive control mechanism that automatically chooses and configures resource management algorithms based on application request patterns and observed system performance. The goal of the Autopilot project is the creation of an infrastructure for building resilient, distributed, and parallel applications. It uses sensors to extract quantitative and qualitative performance data from executing applications, and provides requisite data for decision-making. Artificial Neural Network (ANN) and Hidden Markov Model (HMM) are used for classification. Autopilot uses fuzzy logic to automate the decision making process. The actuators execute the decision by changing parameter values of applications or resource management policies of the underlying system. Active Harmony differs from Autopilot in that it tries to coordinate the use of resources by multiple libraries and applications. Besides, both the instrumentation using sensors and rule-based decision making require more domain knowledge for the program being tuned. Active Harmony tries to provide a tuning mechanism where little or no domain knowledge is required for tuning.

Another approach is application level scheduling. AppLeS [9] allows applications to be informed of the variations in resources and presented with candidate lists of resources to use. In this system, applications are informed of resource changes and provided with a list of available resource sets. Then, each application allocates the resources based upon a customized scheduling to maximize its own performance. The Network Weather Service [34] is used to forecast the network performance and available CPU percentage to AppLeS agents. Active Harmony differs from AppLeS in that we try to optimize resource allocation between multiple libraries and applications, whereas AppLeS lets each application or library adapt itself independently. In addition, by providing a structured interface for applications to disclose their specific preferences, Active Harmony will encourage programmers to think about their needs in terms of options and their characteristics rather than as selecting from specific resource alternatives described by the system.

The GrADS [8] further improves the efforts from Autopilot and AppLeS projects. It tries to simplify distributed heterogeneous computing environment so it is easier for ordinary scientific users to develop, execute, and tune applications on the Grid. Active Harmony can serve as a part of the dynamic optimizer to tailor the reconfigurable applications and systems for good performance with the available resources.

The Odyssey project [24] developed at the University of California at Berkeley focuses on resource awareness at the application level. In this system, each application allocates the resources based upon a customized scheduling to maximize its own performance. Odyssey uses $Fidelity$ as a metric; $Fidelity$ refers to changes in quality of the produced output. The metric is data dependent. For examples, with video, $Fidelity$ might measure

image clarity or compression rate. At all levels of service, $Fidelity$ must be pre-computed and stored at the server. Odyssey only deals with half of the problem. It only handles read operations; it does not concern itself with issues like reintegration, and collaboration with other systems.

The ATLAS (Automatically Tuned Linear Algebra Software) [32], [33] project provides automatically tuned software specialized in linear algebra libraries. They have developed a methodology for the automatic generation of highly efficient basic linear algebra routines for each microprocessor. By using a code generator that probes and searches the system for an optimal set of parameters, it can produce highly optimized matrix multiply routines for a wide range of architectures. The OSKI (Optimized Sparse Kernel Interface) [31] library and the FFTW [18] library also share the spirit of the ATLAS. OSKI is a collection of low-level C primitives that provide automatically tuned computational kernels on sparse matrices, for use in solver libraries and applications. FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data. The difference between those special-purpose libraries and Active Harmony is that our work focuses on general applications that use program libraries rather than that of a specific library. For performance tuning that the problem and the performance model is well-defined, the users should utilize those libraries optimized for the specific functionality. For general purpose performance tuning, Active Harmony should be a good choice.

Prophesy [30] is an infrastructure for analyzing and modeling the performance of parallel and distributed applications. Their work [22] proposed dynamic load balancing for distributed systems like the Grid environment. Instead of the heterogeneity of processors, their framework focuses on the heterogeneity and dynamic load of the network. In our work, Active Harmony improves the performance for scientific programs by tuning the computation and data distributions. In addition, Active Harmony can also tune other parameters such as buffer sizes and number of threads to further improve the performance.

Another project involving load balancing on parallel computers is Zoltan software [14]. Its load-balancing suite provides three classes of parallel partitioning algorithms: geometric bisection, space-filing curve partitioning, and graph partitioning. Zoltan allows an application to switch between partitioning algorithms via a function call. Thus the user may compare different algorithms within a single application easily. This enables users to try several algorithms and find the best ones for their applications. Instead of specific algorithms designed for load-balancing, Active Harmony treats the parameters that affect load-balancing same as other parameters and tune them together to improve the overall performance.

IX. Conclusion

In this paper, we showed that automated performance tuning is useful for scientific libraries and applications to achieve better performance. When tuned, programs can run faster, or achieve higher resolution and use larger data sets in the same amount of time. Another important reason we need automated performance tuning is adaptation. The execution environment may change rapidly and there is no single configuration good for all kinds of environments. Manually tuning may not be a feasible solution since it can be extremely time consuming and the system may have changed again before the manual tuning is completed. Therefore, we need systems such as Active Harmony.

From our experiments, we demonstrated that Active Harmony helps *PETSc* based applications to achieve better load balance and reduce the execution time up to 18%. For *POP*, the tuning results show that there is no single block size good for all topologies. By changing the block size, Active Harmony helps to reduce the execution time up to 15% and by changing the parameter values, the execution time is reduced 16.7% with only 27 iterations. For the *GS2*, we were able to run up to $5.1\times$ faster and thus the developer changed the default data layout configuration.

For the future work, we plan to further analyze the code's behavior for GS2 to better understand the tuning results. To further compare the on-line tuning and off-line tuning, we plan to apply Active Harmony to scientific programs with parameters that can be changed during runtime. The experiment will

compare the results when tuning the parameters on-line and off-line separately.

ACKNOWLEDGMENT

This work was supported in part by NSF award EIA-0080206 and DOE Grants DE-CFC02-01ER25489 and DE-FG02-01ER25510.

The authors thank David Klepacki and Robert Walkup for their reviews and suggestions.

REFERENCES

[1] "libMesh:a c++ framework for the numerical simulation of partial differential equations on serial and parallel platforms." [Online]. Available: http://libmesh.sourceforge.net

[2] B. Appelbe, D. May, S. Quenette, S. Tang, F. Wang, and L. Moresi, "Towards rapid geoscience model development - the snark project," in *Proceedings of 3rd ACES (APEC Cooperation for Earthquake Simulation) Workshop*, 2002.

[3] B. Appelbe, D. May, L. Moresi, S. Quenette, S.-C. Tan, and F. Wang, "Snark - a reusable framwork for geoscience computational models," *Submitted to Pure and Applied Geosciences*.

[4] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc users manual," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 2.1.5, 2004.

[5] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc Web page," 2001. [Online]. Available: http://www.mcs.anl.gov/petsc

[6] S. Balay, V. Eijkhout, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object oriented numerical software libraries," in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Press, 1997, pp. 163–202.

[7] S. J. Benson, L. C. McInnes, J. Moré, and J. Sarich, "TAO user manual (revision 1.7)," Mathematics and Computer Science Division, Argonne National Laboratory, Tech. Rep. ANL/MCS-TM-242, 2004. [Online]. Available: http://www.mcs.anl.gov/tao

[8] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski, "The GrADS Project: Software support for high-level Grid application development," *The International Journal of High Performance Computing Applications*, vol. 15, no. 4, pp. 327–344, 2001.

[9] F. Berman and R. Wolski, "Scheduling from the perspective of the application," in *HPDC '96: Proceedings of the High Performance Distributed Computing (HPDC '96)*. IEEE Computer Society, 1996, p. 100.

[10] K. Bryan, "A numerical method for the study of the circulation of the world ocean," *Journal of Computational Physics*, vol. 135, no. 2, pp. 154–169, 1997.

[11] I.-H. Chung and J. K. Hollingsworth, "Automated Cluster-Based Web Service Performance Tuning," in *HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)*. IEEE Computer Society, 2004, pp. 36–44.

[12] ——, "Using Information from Prior Runs to Improve Automated Tuning Systems," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2004, p. 30.

[13] C. Ţăpuş, I.-H. Chung, and J. K. Hollingsworth, "Active harmony: towards automated performance tuning," in *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. IEEE Computer Society Press, 2002, pp. 1–11.

[14] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan, "Zoltan data management services for parallel dynamic applications," *Computing in Science and Engineering*, vol. 4, no. 2, pp. 90–97, 2002.

[15] W. Dorland, F. Jenko, M. Kotschenreuther, and B. N. Rogers, "Electron temperature gradient turbulence," *Physical Review Letters*, vol. 85, Dec. 2000.

[16] J. K. Dukowicz and R. D. Smith, "Implicit free-surface method for the Bryan-Cox-Semtner ocean model," *Journal of Geophysics Research*, vol. 99, pp. 7991–8014, Apr. 1994.

[17] J. K. Dukowicz, R. D. Smith, and R. C. Malone, "A Reformulation and Implementation of the Bryan-Cox-Semtner Ocean Model on the Connection Machine," *Journal of Atmospheric and Oceanic Technology*, vol. 10, no. 2, pp. 195–208, Apr. 1993.

[18] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[19] J. K. Hollingsworth and P. J. Keleher, "Prediction and adaptation in active harmony," in *HPDC '98: Proceedings of the The Seventh IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society, 1998, p. 180.

[20] P. J. Keleher, J. K. Hollingsworth, and D. Perkovic, "Exposing application alternatives," in *ICDCS '99: Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*. IEEE Computer Society, 1999, p. 384.

[21] M. Kotschenreuther, G. Rewoldt, and W. M. Tang, "Comparison of initial value and eigenvalue codes for kinetic toroidal plasma instabilities," *Computer Physics Communications*, vol. 88, Aug. 1995.

[22] Z. Lan, V. E. Taylor, and G. Bryan, "Dynamic load balancing of samr applications on distributed systems," in *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*. New York, NY, USA: ACM Press, 2001, pp. 36–36.

[23] J. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, 1965.

[24] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile application-aware adaptation for mobility," in *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*. ACM Press, 1997, pp. 276–287.

[25] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed, "Autopilot: Adaptive control of distributed applications," in *HPDC '98: Proceedings of the The Seventh IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society, 1998, p. 172.

[26] R. L. Ribler, H. Simitci, and D. A. Reed, "The autopilot performance-directed adaptive control system," *Future Gener. Comput. Syst.*, vol. 18, no. 1, pp. 175–187, 2001.

[27] W. Scholz, "Magpar: Parallel micromagnetics code." [Online]. Available: http://magnet.atp.tuwien.ac.at/scholz/magpar/

[28] SCIRun Development Team, "SCIRun: A Scientific Computing Problem Solving Environment," 2002. [Online]. Available: http://software.sci.utah.edu/scirun.html

[29] R. D. Smith, J. K. Dukowicz, and R. C. Malone, "Parallel ocean general circulation modeling," in *Proceedings of the eleventh*

annual international conference of the Center for Nonlinear Studies on Experimental mathematics : computational issues in nonlinear science. Elsevier North-Holland, Inc., 1992, pp. 38–61.

[30] V. Taylor, X. Wu, and R. Stevens, "Prophesy: an infrastructure for performance analysis and modeling of parallel and grid applications," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 4, pp. 13–18, 2003.

[31] R. Vuduc, J. W. Demmel, and K. A. Yelick, "OSKI: A library of automatically tuned sparse matrix kernels," in *Proceedings of SciDAC 2005*, ser. Journal of Physics: Conference Series. San Francisco, CA, USA: Institute of Physics Publishing, June 2005.

[32] R. C. Whaley and J. Dongarra, "Automatically tuned linear algebra software," in *SuperComputing 1998: High Performance Networking and Computing*, 1998.

[33] R. C. Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimization of software and the ATLAS project," *Parallel Computing*, vol. 27, no. 1–2, pp. 3–35, 2001.

[34] R. Wolski, "Forecasting network performance to support dynamic scheduling using the network weather service," in *HPDC '97: Proceedings of the 6th International Symposium on High Performance Distributed Computing (HPDC '97)*. IEEE Computer Society, 1997, p. 316.