BLUE WATERS

NCSA

FINAL REPORT

PRAC Topic: Petascale simulations of complex biological behavior in fluctuating environments

NSF Award ID: 0941360

Principal Investigator: Ilias Tagkopoulos, UC Davis

Milestone/Deliverable Description: *Final implementation and final report*

Acceptance Criteria: *The final implementation satisfies the objectives stated in deliverable 1.0: "Report to include description of the implementation and results. Also, it should include user experience, dissemination beyond the development team and documentation."*

Summary

During the support period, the software suite EVE 3.0, a new version of the Evolution in Variable Environments (EVE) framework has been completed. The new parallel model includes an adaptive dynamic load balancer that is implemented based on the analysis of benchmarked prototypes presented in previous and current reports. The performance of the EVE code on the Blue Waters machine is significantly improved, and the code now scales up to 8,000 MPI processes and 128,000 organisms in a population. Every evolutionary experiment requires at least thirty-two independent technical replicate runs for statistics. Therefore, each experiment is potentially scalable up to 256,000 MPI processes.

The combined EVE code incorporates several parallel models: (a) the original model with a static load balancer; (b) a newly-developed adaptive dynamic load balancer with a non-fixed population size option; (c) AMPI compatibility; and (d) a serial version of the code for small local runs. The choice of the optimal model(s) depends on the size of a particular run and on available computational resources.

The C++ source code, compiled binaries for several standard architectures, and the user manual with samples are freely available from the project's website, along with a tutorial on how to use the tool (more information is available at http://www.tagkopolouslab.cs.ucdavis.edu/software). The EVE framework was successfully used recently in microbial evolution research and in the teaching of undergraduate students. More specifically, it was used in both Spring and Fall of 2012 in the ECS 124 "Theory and applications of bioinformatics" class, where approximately fifty undergraduate students used EVE for laboratory assignments related to microbial evolution and population diversity.

1. Parallel models implemented in EVE 3.0

To improve the scalability and performance of the previous version of EVE (v2.1 published in April 2012), we implemented an adaptive dynamic load balancer and semi-global/hierarchical synchronization. While this new parallel model introduces additional overhead on small jobs, it outperforms our original implementation even for a moderate number of processes (> 512 MPI processes) and scales up to about 10,000 MPI processes. This new implementation improves the scalability of the EVE code by a factor of approximately 10x.

1.1. Static load balancer

As described in the previous report, in the original MPI version of EVE the load (population of microorganisms) was divided into chunks (sub-populations) of equal size (Figure 1). Each MPI process was assigned a sub-population of equal, fixed number of organisms. However, as organisms evolve independently, the computational time per organism (and therefore per process) changes over time. The need for frequent synchronization of the population with the environment results in a considerable idling time due to the load imbalance. Additionally, the load imbalance grows with the population size. This resulted in a poor scalability of the original code beyond a few thousand organisms since the probability of the appearance of an unusually large organism, implying more computation within a particular MPI process, grows with the population size.

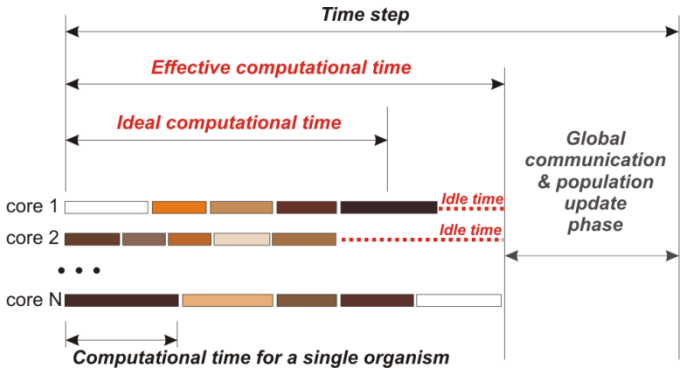

Figure 1. Load imbalance in a static load balancing scheme. Ideal computational time (load) is the average computational time that is required for all MPI processes to update the entire population over one time step in the case of the ideal load balancing. Effective computational time is larger (by about 30% in this cartoon) due to the imperfect load balance in a static scheme with an equal number of organisms per MPI process (core).

Since all organisms in a population are independent and can be updated individually, the parallelization of the code was quite straightforward. However, the original static load balancing model unveiled *two significant problems that affect scalability*: (1) equal load distribution at time step zero was based on the number of organisms, not the computational time required by each organism; (2) load distribution at time step zero was never adjusted during the simulation to accommodate the changes accumulated in an evolving population.

1.2. Adaptive dynamics load balancer

In EVE 3.0 we have implemented an adaptive dynamic load balancer (Figure 2) that addresses both of the problems of static load balancing highlighted above.

Firstly, load balancing is adaptive: the load is distributed among processes to equilibrate the total computational time spent by each process and not the total number of organisms. As a result, the number of organisms per process is not maintained constant anymore. This requires the rearrangement of the original code. In particular, the intra-process communication patterns need to be adjusted and optimized to accommodate the change.

Secondly, load rebalancing is dynamic: rebalancing is performed periodically to minimize the idling time.

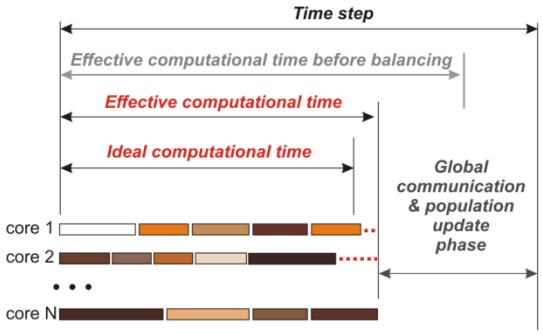

Figure 2. Adaptive load balancer approximates the ideal balance between MPI processes by redistributing organisms according to the required computational time.

The load balancing problem is essentially a *k*-partition problem, or more precisely a bin packing problem: *N* organisms need to be distributed between *k* processes (bins) so that the total computational time is approximately the same on each process. The ideal (minimal) capacity of each bin would be the average computational time per process. Since the population is discrete, in reality the effective capacity of the bins need to be increased relative to the ideal size. To solve the bin packing problem, the load balancer utilizes a basic ‘first fit’ greedy approximation algorithm on a sorted set of organisms. A non-optimal communication pattern does not affect the scalability: as organisms grow and evolve slowly, the balance is maintained except on a few processes and the required rebalancing effort is normally minimal.

In order to accommodate new adaptive dynamic load balancer, a new *population update scheme* was developed for the EVE framework. It allows variable sub-population sizes on each MPI process. The following is the updated population model:

(i) Each process updates its local sub-population. Organisms that grew above the division threshold divide and weak organisms are removed from the sub-population. No inter-process communication in required at this step. In the original model all divisions were non-local to maintain fixed per-process sub-population sizes. That reintroduced additional communication overheads, which are eliminated in the new model.

(ii) Each process creates a list of organisms that need to be sent from it or received from other processes based on the collected computational time. The computational time is approximated with the number of links in the organisms' networks (it was shown in previous reports that this dependence is linear).

(iii) Each process (with MPI rank *i*) establishes pair-wise connections with other processes in the following order [parallel part]:

  (a) Send all organisms that needed to be sent to the processes with a lower rank than *i*. The order: ascending ranks and local organisms' indices.

  (b) Receive all organisms that needed to be received from the processes with a higher rank than *i*. The order: ascending ranks and local organisms' indices.

  (c) Send all organisms that needed to be sent to the processes with a higher rank than *i*. The order: descending ranks and local organisms' indices.

  (d) Receive all organisms that needed to be received from the processes with a lower rank than *i*. The order: descending ranks and local organisms' indices.

1.3. AMPI compatibility

The code was rearranged to become compatible with AMPI (Adaptive MPI) library specifications. In particular, all data structures were made local to each process and sub-routine. AMPI migration is performed with the predetermined frequency. The AMPI addition is compatible with all other parallel models that are implemented in the EVE code. This work was done in collaboration with Ryan Mokos (NCSA). However the AMPI version of the code does not show significant performance improvement over the old version of EVE with the static load balancer. In fact, it scales worse due to the AMPI overhead.

1.4. Semi-global synchronization

All processes are divided into groups to match a physical machine topology. Synchronization is performed independently within each group. Ideally, each group

matches in size and is mapped to a single node or a single blade. Then, using slices of the global MPI world communicator that match the topology of inter-connection between nodes/blades, the collected data is synchronized between groups. The number of inter-group communications is kept to a minimum. The slices are created using custom made MPI communicators that replace the global MPI world communicator.

1.4. Serial version

EVE 3.0 can be compiled as a serial application for local test runs. This option also helps new users to become familiar with the code before using its parallel functionality.

2. Benchmarks and performance

Dynamic adaptive load balancer: Figure 3 depicts the weak scaling of the static and dynamic load balancers. The old parallel model with static load balancer implemented in EVE 2.1 (black line) does not scale beyond 1,000 MPI processes. The newly implemented adaptive dynamic load balancer in EVE 3.0 scales significantly better with reasonable performance up to about 10,000 MPI processes. Certainly scalability is far from ideal (*i.e.*, flat as a function of number of processes), which is the consequence of the biological model: processes need to be synchronized relatively frequently in order to maintain "biological coherence" between organisms evolving on different processes.

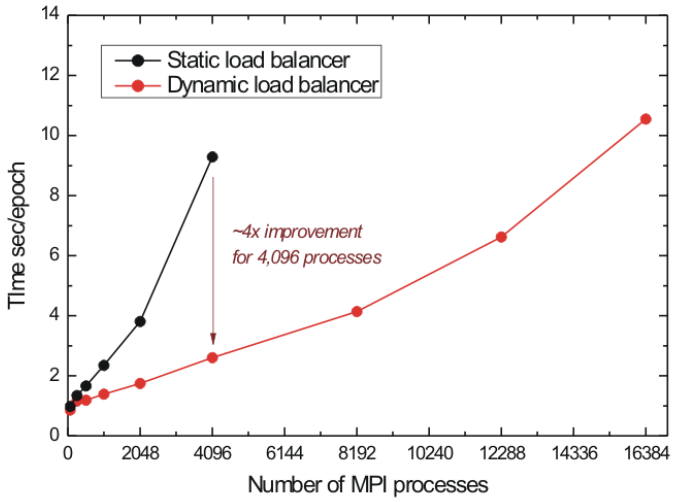

Figure 3. Weak scaling of dynamic (red) and static (black) load balancer models. Eight organisms per process, medium mutation rate AND environment, evolving population.

Figure 4 shows the performance of EVE as a function of per-process load while maintaining a constant number of MPI processes. Theoretically, computational time should increase linearly with the number of organisms per process. For the smaller number of MPI processes (512, left panel) the original static load balancer version of the code actually outperforms dynamic load balancer for a larger number of organisms per process and scales better than the theoretical estimation. The reason for that is the auto-balancing effect of large sub-populations on each process: the load automatically averages between

processes. The dynamic load balancer introduces additional overheads, but nevertheless it scales as expected and even under our theoretical estimation. For larger number of MPI processes (2,048 MPI processes, right panel) communication overheads in old, the static balancer model add to computation time (double it), while the dynamic load balancer, for small average number of organisms per process (up to 16), performs as expected. For larger loads per process (>32 organisms per process) on a large number of MPI processes both balancers perform worse than the expectation with dynamic load balancer being the worst. Therefore for efficient allocation usage no more than 16 organisms per process should be used in large runs.

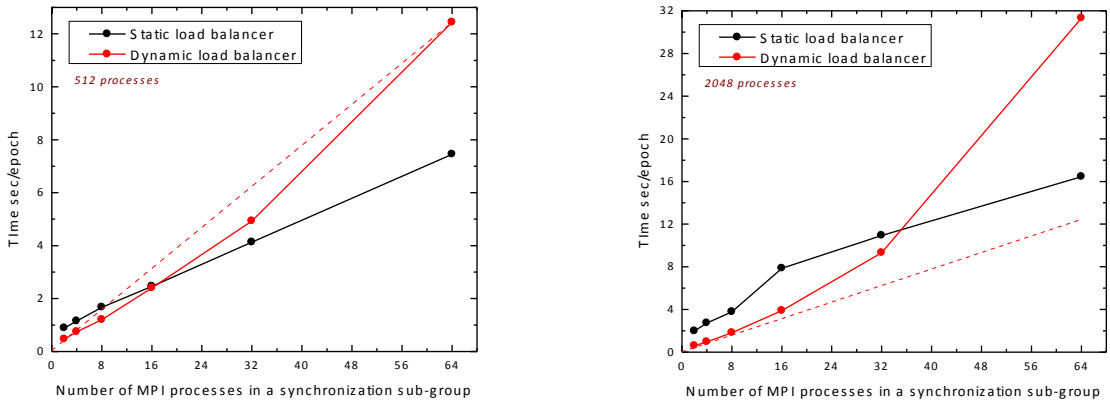

Figure 4. Benchmark results of run times as a function of load per process (*i.e.*, number of organisms per process). Left and right panels show profiling for 512 and 2,048 MPI processes, respectively. The dashed red line shows the ideal/theoretical computational time. Eight organisms per process, medium mutation rate AND environment, evolving population.

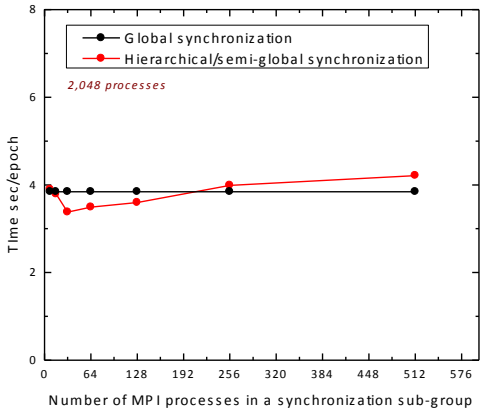

Figure 5. Benchmark results for performance of the semi-global/hierarchical synchronization model (red line) as a function of MPI processes in the lowest level group. The black line shows performance of the code with purely global synchronization. 2,048

MPI processes, eight organisms per process, medium mutation rate AND environment, evolving population.

Semi-global/hierarchical synchronization: Figure 5 shows benchmark results for the semi-global/hierarchical synchronization model against the old, global synchronization model. The improvement is minimal with the best performance at 32 MPI processes per lowest level group. This is not surprising, as it matches the node size on Blue Waters. While the advantage of the new model is not significant, both synchronization models are completely interchangeable. Therefore, the semi-global/hierarchical model with 32 MPI processes per lowest level group should be used in our runs. For machines with poor inter-node communication advantage of the semi-global/hierarchical synchronization with "topology aware" job mapping is much more significant than on Blue Waters.

3. Documentation

Well documented source code in C++, compiled binaries for several standard architectures, a tutorial, and samples are available for download at www.tagkopouloslab.cs.ucdavis.edu. The following distributions are available (each package includes the manual/tutorial and samples):

EVE-v3.0.source.tar.gz	– source code and sample inputs
EVE-v3.0.Linux32.serial.tar.gz	– compiled for Linux 32 bit
EVE-v3.0.Windows32.serial.zip	– compiled for Windows 32 bit
EVE-v3.0.OSX32.serial.tar.gz	– compiled for Mac OSX 32 bit

4. Conclusions and Broader Impacts

EVE is the only microbial evolution simulator that is well parallelized to scale beyond a thousand MPI processes. The framework is universal and it can be appended with any other organism's model (the model is described in the Cell class). EVE is an open source project and it is available for download from our lab's website. EVE has been used successfully for undergraduate training and is part of the core lab assignments for the ECS 124 bioinformatics class (50 students/year).

The work that was performed through this direct PRAC support significantly extended the scalability of EVE in HPC environments in general and the Blue Waters supercomputer specifically. The work that has been performed in EVE will serve as a stepping stone for our next microbial simulator that is organism-specific. Towards that goal, we have created an E. coli simulator, based on a gene expression compendium that we have constructed from both publicly available and proprietary datasets. The simulator is the first that integrates various levels of transcription, translation and metabolic interactions. Our performance analysis and cross-validation testing shows a remarkable performance and predictive capacity in a number of environmental and gene expression perturbation settings. The next step is to use the knowledge and code that we developed through this

award to parallelize our E. coli simulator and incorporate populations of cells. This will allow us to investigate scenarios where emergent behaviors exist on a population level.

References

1. Mozhayskiy V, Tagkopoulos I: Guided evolution of in silico microbial populations in complex environments accelerates evolutionary rates through a step-wise adaptation. *BMC Bioinformatics* 2012, 13(S10):S10.
2. Mozhayskiy V, Tagkopoulos I: Horizontal gene transfer dynamics and distribution of fitness effects during microbial *in silico* evolution. *BMC Bioinformatics* 2012, 13(S10):S13.