Porting and Characterization of GATOR, an Atmospheric Chemical Tracer Model

Balázs Králik and Oleg Zakharov

CS267 Term Paper, Spring 1995

(May 15, 1995)

I. INTRODUCTION

Gator is an atmospheric chemical tracer model. This model is used for studying atmospheric chemistry and has been successfully applied to a number of atmospheric systems including the Los Angeles Basin. The serial version of this code has been developed at AMES/UCLA. The parallel implementation of this code is under development at the Computer Science Department of UC Berkeley (Prof. J. Demmel, Dr. S. Smith).

An important motivation for this research is to provide atmospheric scientists with a valuable tool capable of predicting earth's future climate and using a reasonable amount of the computational resources. This goal requires easily portable parallel code whose performance can be tuned for different parallel architectures. This task is impossible without a performance model for the code. Such model has been developed by Sharon Smith and will be described below. It takes the system performance parameters (such as latency, bandwidth, flop rate) as an input and predicts the computation time on this particular architecture.

The structure of the code is shown in Fig. 1. This figure also demonstrates which parts of the code are described by the existing performance model. The code can be divided into two parts corresponding to different atmospheric processes. The kinetics of the chemical reactions in the atmosphere and some of the physical processes (rainout, dry deposition etc.) are described by ODEs. The correspondent ODE solver is embarrassedly parallel part of the code since it does not require any communication. The deviations from the perfect scaling for this part are only due to the load imbalance as we will see later. The second class of atmospheric processes includes transport phenomena such as diffusion and advection (wind, etc.). The transport processes are divided into the vertical and horizontal transport. The vertical transport does not require any communication and the horizontal transport is the most communication-intensive part of Gator.

Our class project objectives were defined by the general logics of the Gator research project. They are the following:

- Port the existing PVM and CMMD versions of Gator to MPL platform available on SP1 and SP2 parallel computers.
- Benchmark the performance of the SP1 and SP2 to obtain the input parameters for the performance model.
- Using the performance model, analyze the performance of the MPL version of the code and especially its communication-intensive horizontal transport part.
- Find the parameters optimizing the Gator performance for the architecture considered.

II. PORTING GATOR

Gator was ported to the MPL platform by rewriting the macros file in terms of the MPL communication commands. The structure of the code where the architecture specific commands are used to define the platform independent communication macros in the Fortran subroutines makes porting to a new architecture very convenient.

Two versions of the MPL macros were written. In the second version we implemented "strided send" which allows to send several logically independent messages as a single larger message. This option gives us a control over the size of the messages used for the communication. Tuning this size for a given architecture is the main tool for improving the performance of the transport part of the code.

To validate our version of Gator we compared the output data for the gas concentrations with the corresponding results from the previously tested PVM version of the code. The good agreement up to the fourth digit (this is the accuracy for the convergence of the ODE solver) proved that our port was successful.

III. COMMUNICATION PERFORMANCE BENCHMARKS

The motivation for doing our own benchmarks for the communication performance was very poor agreement of our timing results for Gator on SP2 with the numbers obtained using the performance model. We wrote a simple benchmark to measure the latencies and the bandwidths using MPL. Our first results (see Fig. 2) showed that the latencies we were getting were an order of magnitude larger than expected.

This helped us to realize that due to the undocumented feature of system behavior we did not use the High Performance Switch providing low latencies and high bandwidths. Our communication ran over IP instead.

After resolving this problem we started getting the latency and the bandwidth close to the specifications for SP2 and SP1 (see benchmarks in Fig 3 and 4).

The low latencies (an order of magnitude smaller than for the PVM) and high bandwidth for the MPL calls make MPL a communication library of choice for the SP architecture.

IV. THE PERFORMANCE MODEL

The performance model for Gator has been developed by Dr. Sharon Smith. Here we will give a brief description of it since it is important for our performance analysis (for details, see Sharon's paper or Sharon's slides on CS267 Home Page).

The model takes into account only two most time-consuming steps: ODE Solver and Horizontal Transport. For ODE solver the model assumes perfect speedup since this part does not require communication.

$$T_{ODE} = T_{ODE}^{serial}/p$$

The load imbalance can break this approximation to some extent as we will see from our measurements.

The Horizontal Transport is divided into two independent parts: transport along North-South and East-West directions. Each of them in turn divided into the *initial setup phase* when we build (nearly) tridiagonal system of linear equations describing the transport and the *pipeline phase* when we solve this system by pipelining partial results between neighboring processors to overlap communication and computation.

The overall cost of the horizontal transport is

$$T_{HT} = T^{EW}_{setup} + T^{NS}_{setup} + T^{EW}_{solve} + T^{NS}_{solve}.$$

The cost of the setup phase (for the E-W) case is given by the equation

$$T^{EW}_{setup} = 4\left(NS \cdot lat + \frac{\left\lceil \frac{NLAT}{PLAT} \right\rceil \cdot NVS \cdot NBYTES}{bw}\right).$$

Here lat is the latency, bw is the bandwidth, NS is the total number of the setup messages and the numerator in the second term gives the total amount of information sent during the setup phase.

The solve phase time is given by

$$T^{EW}_{solve} = 2 \cdot Nmesg \cdot pxchange \cdot \left(lat + \frac{\left\lceil \frac{NLAT}{PLAT} \right\rceil \cdot NVS \cdot NBYTES}{nr \cdot bw}\right) + \frac{2 \cdot Nmesg \cdot BLONG \cdot \left\lceil \frac{NLAT}{PLAT} \right\rceil \cdot NVS \cdot trans}{m \cdot nr}$$

where

$$Nmesg = min(PLONG, nr) - 1 + max(PLONG, nr+1) \left\lceil \frac{NLONG}{BLONG \cdot PLONG} \right\rceil$$

is the effective number of messages including the cost of filling the pipeline. Without defining all the coefficients in this equations (they can be found in the Sharon's paper) we will give a simplified form for this equation valid for the case of large nr. It reads

$$T^{EW}_{solve} = p(lat \cdot nr + \frac{A}{bw})$$

where nr is the number of messages sent by a given processor, p is some coefficient and A is the total amount of information sent. The parameter nr plays a special role since we can tune it by sending several independent messages. When nr becomes comparable with the pipeline length this simple relation breaks and T^{EW}_{solve} starts increasing when nr decreases. Thus we expect that the optimal value for nr should be of order of the pipeline length equal to the number of different processors in the E-W direction.

V. HORIZONTAL TRANSPORT PERFORMANCE

To tune the performance of the Horizontal Transport routine we timed it for a different number of messages in the pipeline (parameter nr from the previous section).

The execution time for the Horizontal Transport measured on SP1 (with 4 and 8 processors) and SP2 (with 8 and 16 processors) are presented in Fig. 5. The corresponding curves are obtained using the performance model described in the previous section. The measured latencies and bandwidths are used as an input for the model. The latency for SP1 is 43μsec, for SP2 is 60μsec, the bandwidths are 8.5 MB/sec and 32 MB/sec correspondently.

From this graph we see that the overall agreement of our measurements with the model is not very good. First, the model underestimates the execution time for the whole range of message numbers and it does not give the right slope for the large nr region. The agreement is somewhat better for the small numbers of messages. And the model predicts right optimal values for nr in almost all cases.

Both the underestimation of the time and the wrong slope obtained from the model calculation are consistent with the assumption that the effective latency for the code is much larger than the one measured by our benchmarks. To demonstrate that larger latency results in a better agreement with our measurements we calculated the model time using "fake" latencies (220μsec for SP1 and 120μsec for SP2). The curves obtained with this parameters are shown in Fig. 6. Now the agreement with our measurements becomes much better.

We made an attempt to understand what causes these effectively higher latencies. There are several factors which can increase the latency. First, since our implementation of the communication uses non-blocking send (confusingly enough, the send command we are using is called blocking in the MPL manual, but it is blocking only in a sense that it returns only after the sending buffer is cleared. Since the send we use does not wait for a confirmation, we will call it non-blocking), there is a possibility of the network congestion in case we send a a large amount of messages. To check for this possibility we ran a modified benchmark on SP1. Now instead of measuring the round-trip time for one message we send 50 and 100 messages in a row and then receive them back.

Results of this test (see Fig. 7) are somewhat surprising. Instead of increase in latency, in practice, we observe its decrease for the messages of a small size. For the messages with size larger than 200 Bytes the results are practically identical with the results of our previous benchmarks. The possible explanation for such behavior is to assume that the system is smart enough to check if the information it has to send is already in the buffer. In which case some time can be saved by not copying this message from the memory. If this explanation is correct than the critical message size after which we observe increase in the latency is related to the buffer size.

Though our test has shown no indications of the network congestion, this possibility is still not completely excluded since for the real run of the code the load imbalance may result in the inability of some of the processors to receive their messages in time. This is a very important departure from the ideal situation modeled in our test.

To study the effects of the load imbalance on the performance we added several synchronization points to the communication routines. The processors were synchronized between different time steps, after the initial setup phase, and between E-W and N-S transport. These additional synchronization points should improve the load balance since now the calls to the communication-intensive routines happen simolteniously on all the processors. Our timing results show only marginal improvement in the performance due to these changes (47.160 sec vs. 47.169 sec for 4 processors and 40.922 sec vs 42.314 for 8 processors on

SP1 with 96 messages). Of course the synchronization we used can not exclude the load imbalance inside the pipeline itself.

VI. SCALING OF THE MPL VERSION OF GATOR

In this section we consider the scaling properties of Gator and its parts. The scaling of the most time consuming subroutine – ODE solver – is shown in Fig. 8. This scaling is perfect for the case of LA Basin is perfect in the case of Globe we louse some performance due to the load imbalance between the processors. The origin of this imbalance for the block layout used in our calculations is illustrated in Fig. 9. The stiffness of ODE system we have to solve depends on many factors. We need fewer iterations to solve the equations for the winter night conditions than for the summer day. The way around this difficulty is to use the block-cyclic layout providing the static load balance.

The scaling for the horizontal transport part is given in Fig. 10. The optimal value $nr = 8$ has been used for these runs. Though scaling of this part of the code is far from perfect, the time spent in it is an order of magnitude smaller than the computation time for the ODE solver. Thus any further improvement of the scaling for horizontal transport would not result in any significant improvement in the overall performance of the code.

The scaling of the total execution time for Gator is shown in Fig. 11. For the Globe this scaling is mostly defined by the scaling of the ODE solver. Again looking at his graph we conclude that the scaling of the horizontal transport practically is not important for the scaling of Gator for numbers of processors considered. Though we expect that the role of the transport part will increase when we go to 64 or even 128 processor case.

VII. CONCLUSIONS AND FUTURE PLANS

Since we are planning to work on this project over the summer it is important for us to correct our future plans on the basis of the results we were able to obtain up to now.

We demonstrated that the overall performance of Gator can not be improved without improving both parallel and serial performance of the ODE solver at least for the considered number of processors. So it is important to implement the block-cyclic layout to improve the scaling for the ODE solver. It is also important to improve the serial performance for the ODE solver by optimizing this subroutine for the RISC architecture.

Going to the larger number of processors is of great interest since this will change the relative weight of the computation and communication intensive parts of the code. We will study the performance, scaling and agreement with the model for the fine grid on the Globe which requires up to 64 processors.

The improvement of the performance model is also important. We are planning to include into the model the parts of the code which have not been included yet (setup in the Hortran subroutine). We will also try to find the origin of the considerable disagreement between the model and the measured time we observed during the work on our class project.

VIII. INDIVIDUAL CONTRIBUTIONS

The rewriting of the communication macros in term of the MPL communication library has been done by Oleg Zakharov. All the results for the SP2 machine were obtained by Balázs Králik. Both of us contribute to interpreting our results, preparing the posters and the write-up.

ACKNOWLEDGMENTS

We thank Dr. Sharon Smith for her invaluable help at all stages of our project.

FIGURES

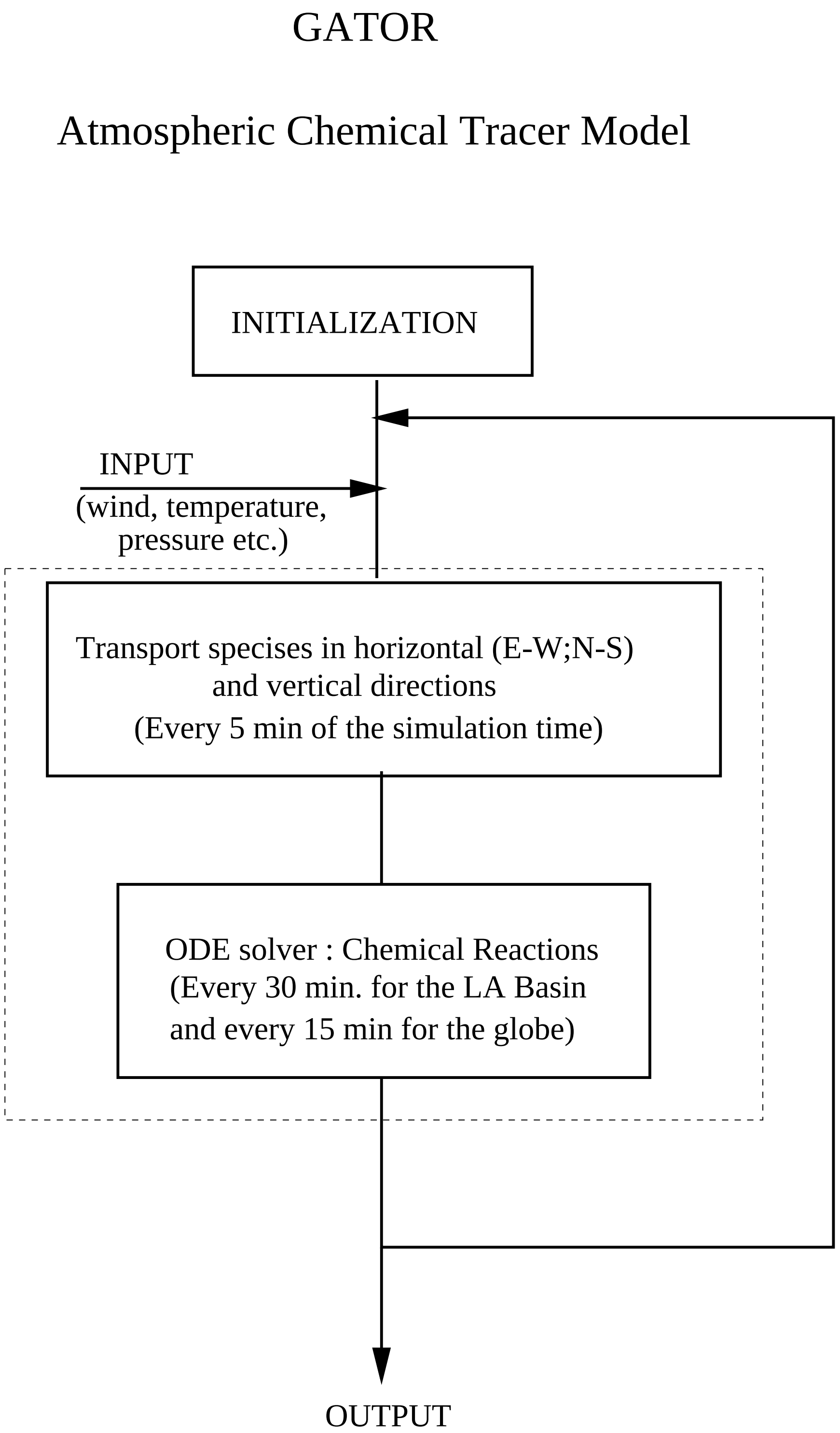


FIG. 1. The structure of Gator

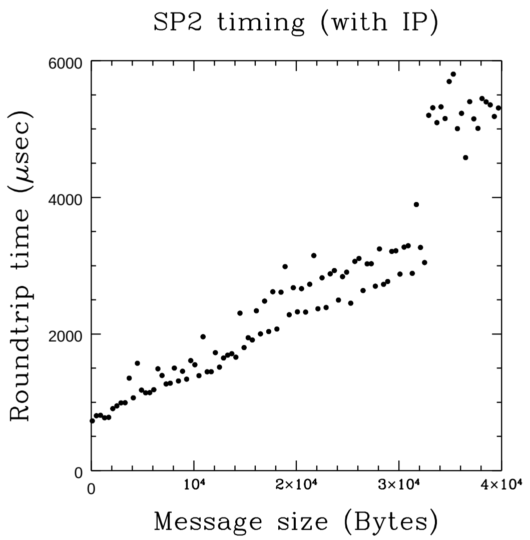


FIG. 2. The communication benchmark for SP2 over IP. The measured latency = 725 μsec, bandwidth = 9.8 MB/sec

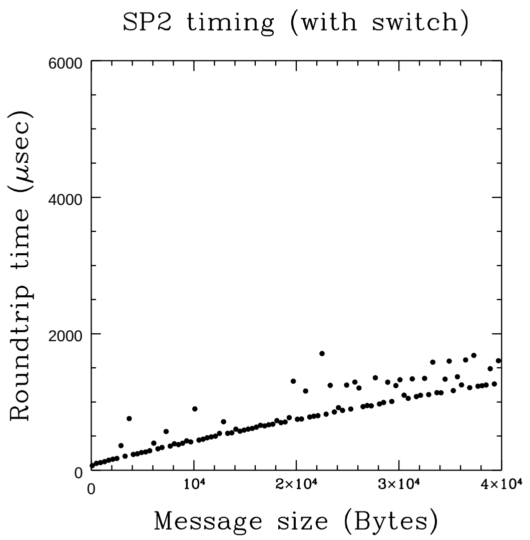
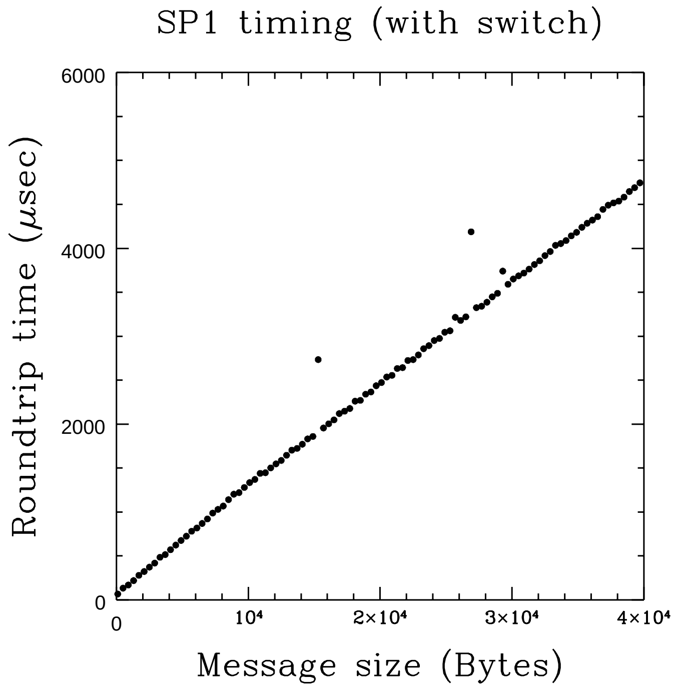


FIG. 3. The communication benchmark for SP2 over the High Performance Switch. The measured latency = 60 μsec, bandwidth = 31.9 MB/sec



.

FIG. 4. The communication benchmark for SP1 over the High Performance Switch. The measured latency = 43 μsec, bandwidth = 8.5 MB/sec

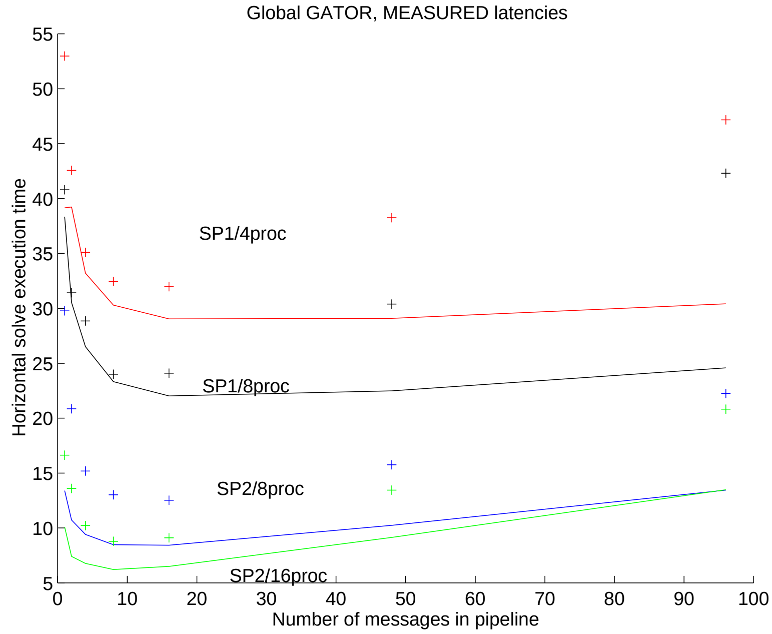


FIG. 5. The measured execution time (dots) for horizontal transport vs the model prediction (lines). The measured latencies and bandwidths are used as the input parameters for the model

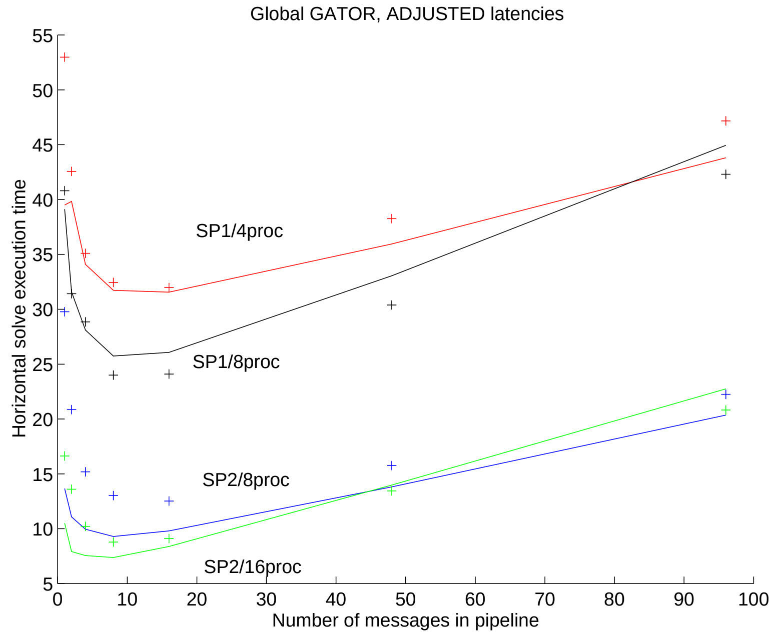
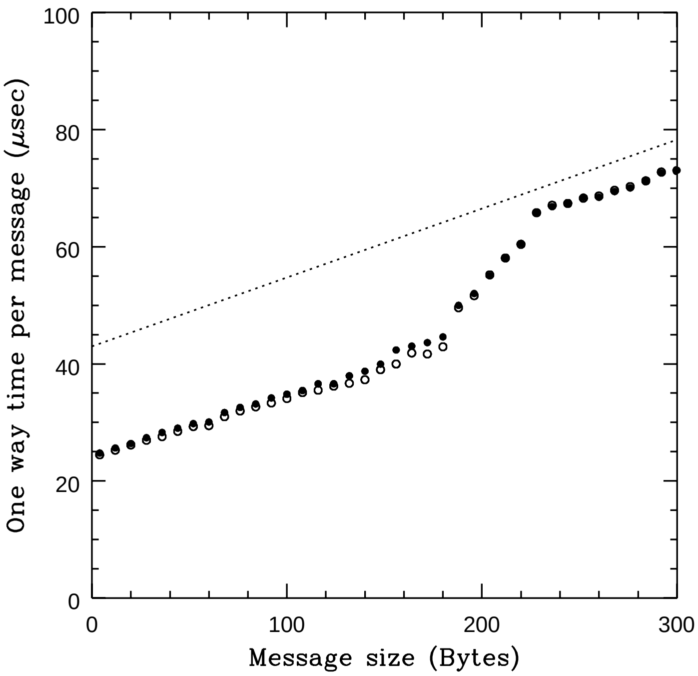


FIG. 6. The measured execution time (dots) for horizontal transport vs the model prediction (lines) with the "fake" larger latencies. The latency of 120μsec is used for SP2, 220μsec for SP1



.

FIG. 7. Modified benchmark for SP1. We send 50 (open dots) and 100 (solid dots) messages in a row. The broken line represents the results of the benchmark when only one message is sent. This plot shows no signs of the network congestion. Instead for small message sizes the latency even becomes lower.

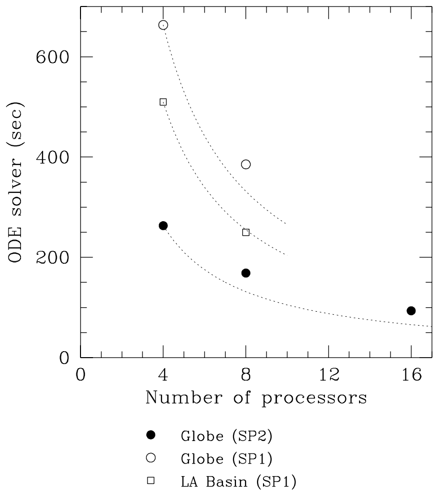


FIG. 8. ODE solver scaling for 8 hour simulation. The broken lines correspond to the perfect scaling

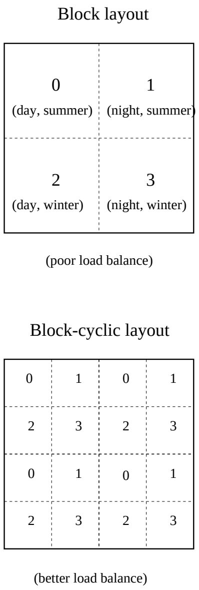


FIG. 9. Block and block-cyclic layouts. The load imbalance for the block layout is illustrated on the example of 4 processors

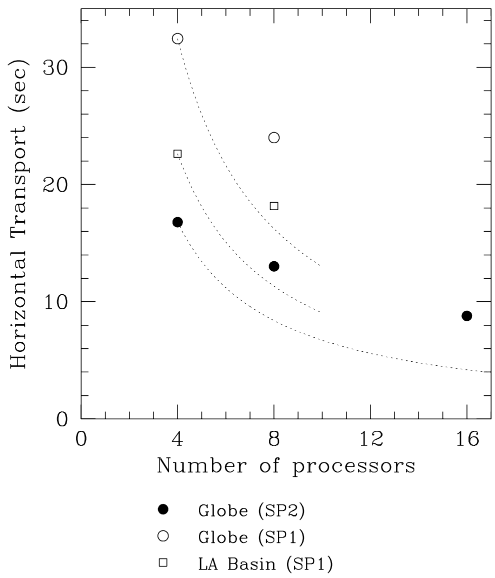


FIG. 10. Horizontal transport scaling for 8 hour simulation. The broken lines correspond to the perfect scaling

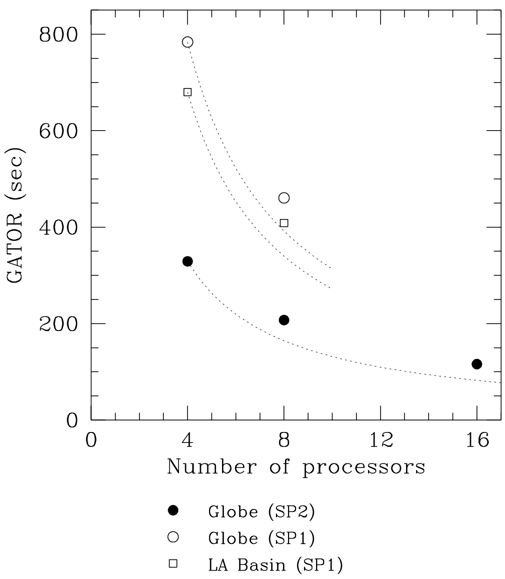


FIG. 11. Overall Gator scaling for 8 hour simulation. The broken lines correspond to the perfect scaling