AEROSPACE REPORT NO.
TOR-2014-00361

Algorithms for Parallelizing AE9/AP9

December 30, 2013

T. Paul O'Brien
Space Science Applications Laboratory
Physical Sciences Laboratories

Prepared for:

Air Force Research Laboratory
Kirtland AFB NM 87117-5776

Contract No. FA8802-09-C-0001

Authorized by: Engineering and Technology Group

PHYSICAL SCIENCES LABORATORIES

The Aerospace Corporation functions as an “architect-engineer” for national security programs, specializing in advanced military space systems. The Corporation's Physical Sciences Laboratories support the effective and timely development and operation of national security systems through scientific research and the application of advanced technology. Vital to the success of the Corporation is the technical staff’s wide-ranging expertise and its ability to stay abreast of new technological developments and program support issues associated with rapidly evolving space systems. Contributing capabilities are provided by these individual organizations:

Electronics and Photonics Laboratory: Microelectronics, VLSI reliability, failure analysis, solid-state device physics, compound semiconductors, radiation effects, infrared and CCD detector devices, data storage and display technologies; lasers and electro-optics, solid-state laser design, micro-optics, optical communications, and fiber-optic sensors; atomic frequency standards, applied laser spectroscopy, laser chemistry, atmospheric propagation and beam control, LIDAR/LADAR remote sensing; solar cell and array testing and evaluation, battery electrochemistry, battery testing and evaluation.

Space Materials Laboratory: Evaluation and characterizations of new materials and processing techniques: metals, alloys, ceramics, polymers, thin films, and composites; development of advanced deposition processes; nondestructive evaluation, component failure analysis and reliability; structural mechanics, fracture mechanics, and stress corrosion; analysis and evaluation of materials at cryogenic and elevated temperatures; launch vehicle fluid mechanics, heat transfer and flight dynamics; aerothermodynamics; chemical and electric propulsion; environmental chemistry; combustion processes; space environment effects on materials, hardening and vulnerability assessment; contamination, thermal and structural control; lubrication and surface phenomena. Microelectromechanical systems (MEMS) for space applications; laser micromachining; laser-surface physical and chemical interactions; micropropulsion; micro- and nanosatellite mission analysis; intelligent microinstruments for monitoring space and launch system environments.

Space Science Applications Laboratory: Magnetospheric, auroral and cosmic-ray physics, wave-particle interactions, magnetospheric plasma waves; atmospheric and ionospheric physics, density and composition of the upper atmosphere, remote sensing using atmospheric radiation; solar physics, infrared astronomy, infrared signature analysis; infrared surveillance, imaging and remote sensing; multispectral and hyperspectral sensor development; data analysis and algorithm development; applications of multispectral and hyperspectral imagery to defense, civil space, commercial, and environmental missions; effects of solar activity, magnetic storms and nuclear explosions on the Earth’s atmosphere, ionosphere and magnetosphere; effects of electromagnetic and particulate radiations on space systems; space instrumentation, design, fabrication and test; environmental chemistry, trace detection; atmospheric chemical reactions, atmospheric optics, light scattering, state-specific chemical reactions, and radiative signatures of missile plumes.

AEROSPACE REPORT NO.
TOR-2014-00361

Algorithms for Parallelizing AE9/AP9

Approved by:

[illegible]

James L. Roeder, Director
Space Sciences Department
Space Science Applications Laboratory
Physical Sciences Laboratories

SC-2350(5666, 13, JS)

Abstract

This report describes algorithms for parallelization of AE9/AP9. Especially for Monte Carlo scenarios, in which the entire mission must be simulated many times, parallelization should afford significant improvements in speed. The simulation can be broken up in the time domain and in the scenario domain. Breaking up the calculation in the time domain requires some care since the “worst-case running average” aggregators (e.g., worst-case one-day average flux) require stitching between time chunks. Parallelizing in the scenario domain requires only that the master processes combine results from the various threads and compute summary statistics (e.g., percentiles). There are additional minor parallelization opportunities as well. This report describes the concept of the various types of parallelization and provides specific algorithms and equations for doing the time-domain stitching of the aggregators.

Acknowledgments

The author acknowledges useful discussions with the AE9/AP9 team.

Contents

Figure

Table

1. Introduction

The AE9/AP9 computer code is far more computationally intensive than any of its predecessors [*Ginet et al.*, 2013]. First, AE9/AP9 represents the environment in terms of directional fluxes, which, for most applications, it must integrate numerically to compute an omnidirectional flux. In comparison, AE8/AP8 only had to interpolate omnidirectional fluxes to the spacecraft location. This adds a factor of ~10 to the number of fluxes that must be calculated in AE9/AP9. Second, AE9/AP9 uses a more sophisticated drift shell magnetic coordinate system to represent the fluxes, and those coordinates take longer to compute than the field-line coordinates used by AE8/AP8. Third, in order to provide confidence intervals, AE9/AP9 provides a set of scenarios, with each scenario having a perturbed statistical map from which the fluxes are drawn, and there being 40 or more scenarios. Finally, in order to represent dynamics for worst-case calculations, it is necessary to simulate the entire mission under consideration, rather than just a few representative orbits [*O'Brien*, 2007]. Together, all of these features of AE9/AP9 suggest the need for parallelization.

Individual AE9/AP9 scenarios are independent, thus the scenario is a natural dimension along which to parallelize. Also, AE9/AP9 was designed to permit parallelization in the time domain through the use of an "epoch time" input that initializes the random number generator for a specified start date and then "winds-forward" the state variables to the start of the time chunk to ensure that all chunks computed separately are identical to what they would have been had they been computed serially. Additionally, the random number seed is specific to each scenario, so that a scenario will be the same whether it is run in isolation or with many other scenarios. Finally, a single "run" of AE9/AP9 may be asked to compute a number of useful quantities, which we call aggregators. An example aggregator is the total dose behind a range of shielding depths, or the worst-case 24-hour average electron flux. Since the aggregators are independent of each other, they afford another possibility for parallelization.

2. Algorithm overview

Figure 1 show an overview of a nearly maximal parallelization strategy, i.e., one that minimizes redundant calculations while assuming a low cost for inter-thread communication (e.g., on a shared-memory system). We assume a master thread that launches several slave threads in a sequence of fork and join operations, until the final results are written to the file system.

2.1 Steps 1 and 2—Setup

In step 1 in Figure 1, the master thread ingests user input. The user specifies which model to run, what kind of run to do, the ephemeris for the run, and what aggregators to use. The master thread then breaks up the ephemeris into time chunks that are passed to slave threads for the computation of the projection weights. Projection weights require magnetic coordinates for each time point; then, those coordinates are used to compute the matrices that project the global model flux map onto the spacecraft trajectory. The calculation of coordinates is independent for each time point, so it is natural to split up the calculation along the time domain. The coordinates are re-used for every scenario, so there is no need (yet) to split along the scenario dimension. Step 2 has each slave thread computing coordinates and weights for one time chunk, then passing those results back to the master thread. The magnetic coordinate calculation takes longer depending on the location of the ephemeris point. Therefore, step 2 could benefit substantially from load balancing (although the value of load balancing diminishes if the time chunk is longer than a few orbits).

2.2 Steps 3 and 4—Computing Flux

In step 3 in Figure 1, the master thread again breaks the problem up in the time domain (not necessarily in the same time chunks used by steps 1 and 2) in preparation for the flux calculation. Because at this point in the processing the aggregators require the raw time series of flux at the spacecraft, the master thread must also prepare the aggregators to operate on each time chunk. Finally, the master thread can set up the calculation for each scenario (with its own instances of the aggregators) to run independently. In step 4, each slave thread computes the flux and runs the aggregator for one time chunk and one scenario. The aggregator partial results and ghost data (but not necessarily the full flux time series) are reported back to the master thread.

Because the magnetic coordinate data is very large, it is much more efficient on non-shared-memory machines to send the run parameters for steps 3 and 4 along with the dispatch from step 1 to 2 so that there is no need to return information to the master thread between steps 2 and 4. In this approach, step 3 would occur on each of the slave threads from step 2, and each slave thread would fork multiple subthreads (up to one per scenario) for step 4.

Is it even useful to split in time and scenario? The extra overhead associated with splitting along two dimensions simultaneously may not be worth the trouble if one can split arbitrarily finely in either of the dimensions. In fact, once one has committed to splitting in the time dimension, there is not necessarily anything to be gained by also splitting along the scenario dimension in step 4. Conversely,

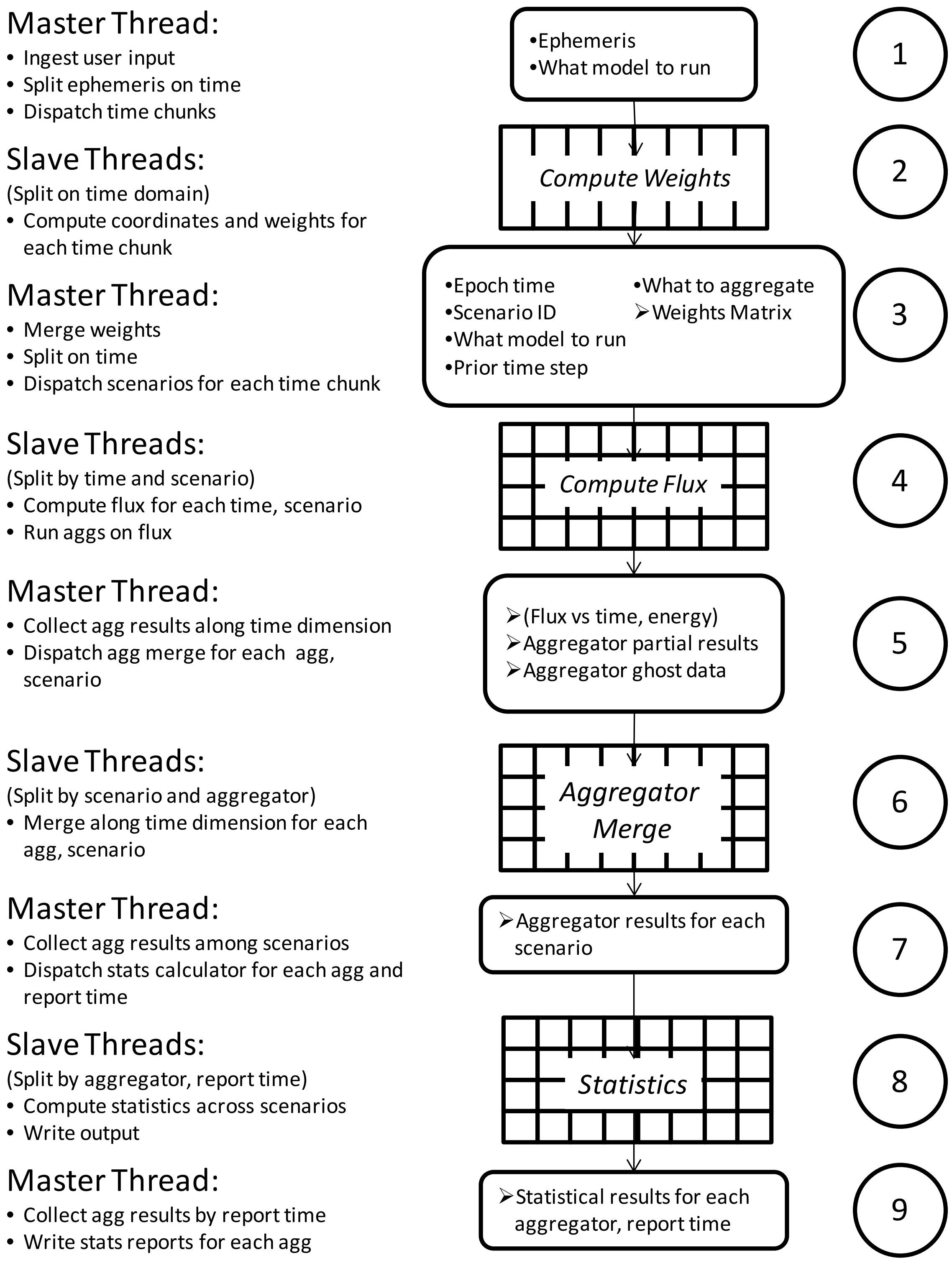
Master Thread:
• Ingest user input
• Split ephemeris on time
• Dispatch time chunks
•Ephemeris
•What model to run
1
Slave Threads:
(Split on time domain)
• Compute coordinates and weights for each time chunk
Compute Weights
2
Master Thread:
• Merge weights
• Split on time
• Dispatch scenarios for each time chunk
•Epoch time
•Scenario ID
•What model to run
•Prior time step
•What to aggregate
➢Weights Matrix
3
Slave Threads:
(Split by time and scenario)
• Compute flux for each time, scenario
• Run aggs on flux
Compute Flux
4
Master Thread:
• Collect agg results along time dimension
• Dispatch agg merge for each agg, scenario
➢(Flux vs time, energy)
➢Aggregator partial results
➢Aggregator ghost data
5
Slave Threads:
(Split by scenario and aggregator)
• Merge along time dimension for each agg, scenario
Aggregator Merge
6
Master Thread:
• Collect agg results among scenarios
• Dispatch stats calculator for each agg and report time
➢Aggregator results for each scenario
7
Slave Threads:
(Split by aggregator, report time)
• Compute statistics across scenarios
• Write output
Statistics
8
Master Thread:
• Collect agg results by report time
• Write stats reports for each agg
➢Statistical results for each aggregator, report time
9

Figure 1. Overview strategy for maximal parallelization

splitting only in the scenario dimension is less useful because one cannot split to more threads than there are scenarios. On a shared-memory system with only 4–16 processors, and a typical set of 40 or more scenarios to run, this is not an issue. However, on a cluster, with potentially hundreds of threads and little shared memory, this distinction is very important. We will see that splitting in the time domain does have its costs, but avoiding them to split only in the scenario domain would lock us into a parallelization scheme that cannot scale up to use the full available computational resources.

2.3 Steps 5 and 6—Aggregation

In step 5, the master thread collects the aggregator results along the time dimension in preparation for merging them together. Then, the master dispatches one aggregator and one scenario to each slave thread for merging the aggregators' results along the time domain. Section 3 describes the aggregator partial results, ghost data, and merging algorithms. Step 6 is broken up by scenario and aggregator. Again, a 2-D split is depicted, but this time neither dimension of the 2-D split can be divided into arbitrarily small pieces: the user defines the number of scenarios and aggregators, so splitting in two dimensions increases the utilization of available processors. It is noteworthy that different aggregators involve different levels of complexity in their merge step (step 6). So steps 5 and 6 are an important place for load balancing. At the completion of step 6, the slave threads return final results for each aggregator for each scenario.

2.4 Steps 7, 8, and 9—Summary Statistics

In step 7, the master thread collects the aggregator results and groups all the scenario results for a particular aggregator. It then dispatches each aggregator to a slave thread for summary statistics (such as percentiles), which are computed by analyzing the distribution of results across scenarios. Because the user may have requested several intermediate reporting times as well as the final report from each aggregator, it is possible to split step 8 by both the aggregator and the report time. In step 8, each slave thread computes the summary statistics for each report time and returns them to the master thread.

Finally, in step 9, the master thread collects the reports by aggregator, and produces one time-ordered file of summary statistics for each aggregator.

2.5 Prioritizing Parallelization

We have a mixed experience with regards to whether it takes longer to do step 4 or step 5. For a fluence/dose type run, step 4 is clearly the longest step because the aggregators are trivially simple. However, when worst-case aggregators are included, step 6 can actually exceed step 4. When a very long time interval is to be simulated, step 2 can become significant. When frequent (e.g., daily) reporting is requested, step 8 can become significant. However, for most of the cases we have run, it is steps 4 and 6 that dominate the calculation.

3. Algorithms for Parallelizing Aggregators

AE9/AP9 supports several different aggregators. They are listed in Table 1. All the aggregators are based on the time series *j(E,t)*, which is the flux versus energy at a specific time. The *dose* and *doserate* aggregators rely on ShielDose2 [*Seltzer*, 1994], *S(J(E,t);d)*, and so work only with omnidirectional differential (per unit energy) flux. The other aggregators can work with any kind of flux.

We will assume that the time domain is divided into N_c time chunks, each of which spans a duration T_c, and covers times from t_{c-1} to t_c, or discrete time steps $t_{1+k_{c-1}}$ to t_{k_c}. Time tags are elapsed time, so continuous time starts at t=0 and the first discrete time tag is t_1=0. The subscript *c* will be used to represent information specific to time chunk *c*. Reports are generated at times designated t_r or discrete time steps t_{k_r}.Both *c* and *r* start at 1. In some of the equations below, we will see references to k_{c-1}; for the edge case when *c*=1, we take $k_{c-1}=k_0=0$; thus, a reference to $t_{1+k_{c-1}}$ for $c=1$ is actually a reference to t_1.

We will work through each of the aggregators, in turn, explaining how it can be parallelized over time chunks. Because the flux time series itself can be enormous (e.g., 20 floating-point numbers every 10 seconds of simulated time, for many scenarios and many simulated years), we have striven to parallelize the aggregators in a way that requires minimal passing of raw flux time series data between the slaves and the master thread. Thus, the aggregator may return partial results for the time chunk as well as ghost data that is needed to merge the partial results from multiple time chunks, but it is assumed the flux time series itself will never be passed between threads, except as contained in aggregator ghost data.

Table 1. AE9/AP9 Aggregators

Name	Equation	Description
Fluence	$J(E,t) = \int_0^t j(E,t')dt'$	Time integral of flux along orbit versus energy since start of mission
Mean	$\bar{J}(E,t) = J(E,t)/t$	Time average of flux along orbit versus energy since start of mission
Dose	$D(d,t) = \int_0^\infty G(d,E)J(E,t)dE = S(J(E,t);d)$	Total dose since start of mission
Boxcar	$\bar{J}_{\max}(E,t;\tau) = \max_{\tau \le t' \le t} \bar{J}(E,t';\tau)$ $\bar{J}(E,t;\tau) = \frac{1}{\tau}\int_{t-\tau}^{t} j(E,t')dt'$	Worst case box-car average flux (time window τ) since start of mission

Name	Equation	Description
Expavg	$\tilde{j}_{\max}(E,t;\tau) = \max_{t}\left[\frac{1}{\tau}\int_0^t \exp\left(\frac{t-t'}{\tau}\right) j(E,t')dt'\right]$	Worst case exponential average flux (time window τ) since start of mission (simulates RC circuit for internal charging analysis)
Doserate	$\dot{D}(d,t;\tau) = \int_0^\infty G(d,E)\bar{j}(E,t;\tau)dE = S(\bar{j}(E,t;\tau);d)$	Dose rate averaged over preceding time interval (τ)

3.1 The Fluence Aggregator

We begin with the *fluence* aggregator, and recast it as a discrete sum:

$$J(E_i,t_k) = \sum_{k'=1}^{k} j(E_i,t_{k'})\Delta t_{k'} \tag{1}$$

$$\Delta t_k = \begin{cases} t_2 - t_1 & k=1 \\ t_{N_k} - t_{N_k-1} & k = N_k \\ \frac{t_{k+1}-t_{k-1}}{2} & \text{otherwise} \end{cases}. \tag{2}$$

We have defined Δt_k under the assumption that each time point represents a time bin, rather than an entry in a time grid. Thus, to compute fluence for a day, the first time point would be at midnight, and the last time point would be *just before* midnight on the next day.

The fluence calculation can be parallelized by means of the partial fluence $J_c(E_i,t_r)$ accumulated within time chunk *c*, and the final chunk partial fluence $J_c(E_i,t_{k_c})$:

$$J_c(E_i,t_k) = \sum_{k'=1+k_{c-1}}^{k_c} j(E_i,t_k)\Delta t_k. \tag{3}$$

We note that to compute Δt_k the aggregator must know the time tags immediately before and after the time chunk. That is, it would need to know $t_{k_{c-1}}$and t_{1+k_c}. The aggregator would also need to know N_k.

Ghost data and partial results for the *fluence* aggregator are thus:

- Input

  t_k for k in [k_{c-1},1+k_c]

  k_c, N_k

- Output

  $J_c(E_i,t_{k_r})$ for k_r in [1+k_{c-1},k_c]

  $J_c(E_i,t_{k_c})$

Stitching together *fluence* for time chunks is simple:

$$J\left(E_i, t_{k_r}\right) = J_c\left(E_i, t_{k_r}\right) + \sum_{c'=1}^{c-1} J_{c'}\left(E_i, t_{k_{c'}}\right). \tag{4}$$

That is, the reported fluence at t_{k_r} is the partial fluence $J_c\left(E, t_{k_r}\right)$ for the time chunk containing k_r plus the sum of the final partial fluences of the previous time chunks.

3.2 The Mean Aggregator

The *mean* aggregator in discrete form is:

$$\bar{J}\left(E_i, t_{k_r}\right) = \begin{cases} J(E_i, t_{k_r})/t_2 & k_r = 1 \\ J(E_i, t_{k_r})/t_{k_r} & \text{otherwise} \end{cases} \tag{5}$$

Parallelizing the *mean* aggregator builds on the *fluence* aggregator, using the same ghost data and partial results, with Eq. (5) applied after Eq. (4).

3.3 The Dose Aggregator

The *dose* aggregator in discrete form is:

$$D\left(d, t_{k_r}\right) = S\left(J\left(E_i, t_{k_r}\right); d\right). \tag{6}$$

As with the *mean* aggregator, parallelizing the *dose* aggregator builds on the *fluence* aggregator, using the same ghost data and partial results, with Eq. (6) applied after Eq. (4).

3.4 The Boxcar Aggregator

The boxcar aggregator is the maximum-to-date of a windowed running average, with window length τ. The discrete representation of the *boxcar* aggregator is:

$$\bar{J}_{\max}(E_i, t_k; \tau) = \max_{\tau \le t_{k'} \le t_k} \bar{J}(E_i, t_{k'}; \tau) \tag{7}$$

$$\bar{J}(E_i, t_k; \tau) = \frac{1}{\tau}\left(J(E_i, t_k) - J(E_i, t_k - \tau)\right). \tag{8}$$

We note that the *boxcar* worst case is not defined for $t < \tau$, because averages over intervals shorter than τ can easily exceed the meaningful worst case for averages of length τ. The *boxcar* and *doserate* aggregators both rely on a box-car running average, $\bar{J}(E, t; \tau)$ with time window τ. That running average is, in turn, computed from the fluence at time t_k minus the fluence at time $t_k - \tau$ in the past. It is best to compute the past fluence $J(E_i, t_k - \tau)$ by interpolation in time.

Ghost data and partial results for the *boxcar* aggregator are thus:

- Input

t_k for k in $[k_{c-1}, 1+k_c]$

k_c, N_k

- Output

  $\bar{J}_{\max,c}(E_i, t_{k_r}; \tau)$ for t_{k_r} in $[t_{1+k_{c-1}} + \tau, t_{k_c}]$

  $\bar{J}_{\max,c}(E_i, t_{k_c}; \tau)$

  $J_c(E_i, t_k)$ for t_k in $[t_{1+k_{c-1}}, t_{k_{c-1}} + \tau]$ or $[t_{k_c} - \tau, t_{k_c}]$

The partial result $\bar{J}_{\max,c}$ is given by:

$$\bar{J}_{\max,c}(E_i, t_k; \tau) = \max_{t_{1+k_{c-1}} + \tau \le t_{k'} \le t_k} \bar{J}(E_i, t_{k'}; \tau). \tag{9}$$

That is, it is the maximum starting τ after the beginning of time chunk c rather than at t=0.

First, we stitch $\bar{J}$ for the edge window cases, i.e., when $t_{1+k_{c-1}} \le t_{k_r} < t_{1+k_{c-1}} + \tau$ for any c>1 using Eq. (4) then Eq. (8). Equation (4) is not needed when $T_c > \tau$, and one can simply replace $J(E_i, t_k)$ in Eq. (8) with $J_c(E_i, t_k)$. We can then compute $\bar{J}_{\max,c}(E_i, t_{k_r}; \tau)$ for all t_{k_r} in the edge window:

$$\bar{J}_{\max,c}(E_i, t_{k_r}; \tau) = \max_{t_{1+k_{c-1}} - \tau \le t_{k'} \le t_{k_r}} \bar{J}(E_i, t_{k'}; \tau). \tag{10}$$

Finally, we revise all the partial maxima $\bar{J}_{\max,c}(E_i, t_{k_r}; \tau)$ into $\bar{J}_{\max}(E_i, t_{k_r}; \tau)$ by taking the maximum over the previous partial maxima and also the previous end-of-chunk maxima:

$$\bar{J}_{\max}(E_i, t_{k_r}; \tau) = \max\left[\max_{c' < c} \bar{J}_{\max,c}(E_i, t_{k_{c'}}; \tau), \max_{r' \le r} \bar{J}_{\max,c}\left(E_i, t_{k_{r'}}; \tau\right)\right]. \tag{11}$$

3.5 The Doserate Aggregator

The *doserate* aggregator has a very simple special case when the reporting period exactly matches the averaging window (i.e., in continuous time $t_r = t_{r-1} + \tau$), the time step is constant, and the reporting time is an integer multiple of the time step. This special case can be addressed by simple post-processing of the *dose* aggregator:

$$\dot{D}(d, t_{k_r}; \tau) = \begin{cases} \frac{D_d(d, t_{k_r})}{\tau} & r = 1 \\ \frac{D_d(d, t_{k_r}) - D_d(d, t_{k_{r-1}})}{\tau} & r > 1 \end{cases} \tag{12}$$

In a serial implementation of the *doserate* aggregator, if the time window τ exactly matches the reporting period, then there is no need for ghost data. The accumulator for $\bar{J}(E, t; \tau)$ can simply be reset after each report is stored. However, in a parallel implementation, where time windows may involve consecutive time chunks, at least some ghost data is necessary anyway; thus, the parallel implementation also enables the unlikely case of a time window that is not the same as the reporting window; it also addresses the likelihood that the time step is uneven, or that the reporting time is not an integer multiple of the time step.

For parallel implementation, the *doserate* aggregator in discrete form is:

$$\dot{D}\left(d, t_{k_r}; \tau\right) = S\left(\bar{j}\left(E_i, t_{k_r}; \tau\right); d\right) \tag{13}$$

The parallel implementation of the *doserate* aggregator is similar to the *boxcar* aggregator, except no running maximum of $\bar{j}\left(E_i, t_{k_r}; \tau\right)$ is computed, and there is the added post-processing through *S*.

Ghost data and partial results for the *doserate* aggregator are thus:

- Input

  t_k for k in $[k_{c-1}, 1+k_c]$

  k_c, N_k

- Output

  $\dot{D}(d, t_{k_r}; d)$ for t_{k_r} in $[t_{1+k_{c-1}} + \tau, t_{k_c}]$

  $J_c(E_i, t_k)$ t_k in $[t_{1+k_{c-1}}, t_{k_{c-1}} + \tau]$ or $[t_{k_c} - \tau, t_{k_c}]$

As in the boxcar, we stitch $\bar{j}$ for the edge window cases, i.e., when $\left|t_{k_r} - t_{k_c}\right| < \tau$ for any $1 \leq c < N_c$ using Eq. (4) then Eq. (8) as for the *boxcar*. Then we compute $\dot{D}\left(d, t_{k_r}; \tau\right)$ for the edge cases using Eq. (13).

3.6 The Expavg Aggregator

The *expavg* aggregator is potentially the most challenging to break up in time. This is because the running average is an *infinite impulse response moving average filter*; at every time step, it responds to all prior fluxes.

The *expavg* aggregator in discrete form is:

$$\tilde{j}_{\max}(E_i, t_k; \tau) = \max_{t_{k'}} \tilde{j}(E_i, t_{k'}; \tau) \tag{14}$$

$$\tilde{j}(E_i, t_k; \tau) = \left[1 - \exp\left(\frac{t_{k-1} - t_k}{\tau}\right)\right] \mathrm{j}(E_i, t_k) + \exp\left(\frac{t_{k-1} - t_k}{\tau}\right) \tilde{j}(E_i, t_{k-1}; \tau)$$

$$= \sum_{k'=1}^{k} \mathrm{j}(E_i, t_{k'}) \left[1 - \exp\left(\frac{t_{k'-1} - t_{k'}}{\tau}\right)\right] \exp\left(\frac{t_{k'} - t_k}{\tau}\right)$$

$$= \exp\left(\frac{t_{k_{c-1}} - t_k}{\tau}\right) \tilde{j}\left(E_i, t_{k_{c-1}}; \tau\right) + \sum_{k'=1+k_{c-1}}^{k} \mathrm{j}(E_i, t_{k'}) \left[1 - \exp\left(\frac{t_{k'-1} - t_{k'}}{\tau}\right)\right] \exp\left(\frac{t_{k'} - t_k}{\tau}\right) \tag{15}$$

The third form of Eq. (15) allows us to stitch together consecutive chunks. We could define a partial running average as

$$\tilde{j}_c(E_i, t_k; \tau) = \sum_{k'=1+k_{c-1}}^{k} \mathrm{j}(E_i, t_k)\left[1 - \exp\left(\frac{t_{k'-1} - t_{k'}}{\tau}\right)\right]\exp\left(\frac{t_{k'} - t_k}{\tau}\right) \tag{16}$$

We could have each slave thread return the entire partial moving average time series $\tilde{j}_c(E_i, t_k; \tau)$, and then compute the whole series of moving averages:

$$\tilde{j}(E_i, t_k; \tau) = \tilde{j}_c(E_i, t_k; \tau) + \sum_{\mathrm{c}'<c}\tilde{j}_{c'}\left(E_i, t_{k_{c'-1}}; \tau\right)\exp\left(\frac{t_{k_{c'-1}} - t_k}{\tau}\right) \tag{17}$$

We could then apply Eq. (14) to compute $\tilde{j}_{\max}(E_i, t_k; \tau)$.

However, we would like to avoid the need to pass the entire time series $\tilde{j}_c(E_i, t_k; \tau)$ between threads because it can amount to a lot of data. Instead, we will have the slave pass back partial results $\tilde{j}_{\max,c}\left(E_i, t_{k_r}; \tau\right)$ at the reporting times and ghost data $\tilde{j}(E_i, t_k; \tau)$ for a window that is several (e.g., N_τ=10) τ long. For raw fluxes that vary over 5 orders of magnitude, a ghost window of 10τ ensures that the contribution of anything prior to the ghost window is negligible. Such a strategy degenerates to passing the entire series of $\tilde{j}_{\max,c}(E_i, t_k; \tau)$ when N_τ τ is longer than the time chunk T_c. A typical internal charging specification employs a τ of 1 day, but special cases may employ a τ of 6 months or longer.

Ghost data and partial results for the e*xpavg* aggregator are thus:

- Input

  t_k for k in [k_{c-1},k_c]

  k_c, N_k

- Output

  $\tilde{j}_{\max,c}\left(E_i, t_{k_r}; \tau\right)$ for t_{k_r} in $[t_{1+k_{c-1}} + N_\tau\tau, t_{k_c}]$

  $\tilde{j}_c\left(E_i, t_{k_c}; \tau\right)$

  $\tilde{j}_c(E_i, t_k; \tau)$ for t_k in $[t_{1+k_{c-1}}, \min\,(t_{k_c}, t_{1+k_{c-1}} + N_\tau\tau)]$

The partial result $\tilde{j}_{\max,c}\left(E_i, t_{k_r}; \tau\right)$ is given by:

$$\tilde{j}_{\max,c}(E_i, t_k; \tau) = \max_{t_{1+k_{c-1}} \le \mathrm{t}_{\mathrm{k}'} \le t_{k_c}} \tilde{j}(E_i, t_{k'}; \tau) \tag{18}$$

We can compute $\tilde{j}(E_i, t_k; \tau)$ for t_k in the edge window $[t_{1+k_{c-1}}, \min\,(t_{k_c}, t_{1+k_{c-1}} + N_\tau\tau)]$ by adding in the last value from the prior time chunks according to Eq. (17). From this, we can compute $\tilde{j}_{\max}\left(E_i, t_{k_r}; \tau\right)$ for all t_{k_r} in the edge window. Then, for all t_{k_r} outside the edge window, we simply add on the prior end-of-chunk values $\tilde{j}_c\left(E_i, t_{k_c}; \tau\right)$ with a proper exponential decay factor:

$$\tilde{j}_{\max}(E_i, t_{k_r}; \tau) = \max[\tilde{j}_{\max}(E_i, t_{k_{r-1}}; \tau), \tilde{j}'_{\max}(E_i, t_{k_r}; \tau)] \tag{19}$$

$$\tilde{j}'_{\max}(E_i, t_{k_r}; \tau) \approx \begin{cases} \tilde{j}_{max}(E_i, t_{k_r}; \tau) & 0 \le t_{k_r} - t_{1+k_{c-1}} \le N_\tau \tau \\ \tilde{j}_{max,c}(E_i, t_{k_r}; \tau) + \sum_{c'=1}^{c-1} \tilde{j}_{c'}(E_i, t_{k_{c'-1}}; \tau) \exp\left(\frac{t_{k_{c'-1}} - t_k}{\tau}\right) & \text{otherwise} \end{cases} \tag{20}$$

We are thus assuming that by the time we get to $t_{k_r} > t_{1+k_{c-1}} + N_\tau \tau$, all contributions to $\tilde{j}(E_i, t_k; \tau)$ from prior time chunks are negligible. In Eq. (19), we take the maximum-to-date to ensure that a maximum from a prior time chunk or a reconstructed maximum earlier in the same time chunk appropriately replaces the partial result maximum.

We have conducted several numerical experiments using the algorithms described here and have confirmed that for a log-normally varying time series that spans 5 orders of magnitude, this approach works to 10^{-13} maximum relative error (a couple orders of magnitude larger than the floating-point error because we are doing sums of many variables, but well below the needs of satellite design, for which relative error 10^{-2} is more than adequate). We have also confirmed that adding in serial correlation (lag correlation) to the underlying data does not cause the algorithm to fail.

4. Summary

We have described the overall approach to parallelizing large AE9/AP9 calculations. We have discussed different strategies for shared-memory and non-shared-memory systems (i.e., clusters). We have also defined the various aggregators built into AE9/AP9 and how to break those aggregators up over time.

In only one case, namely, the *expavg* aggregator, which approximates an RC circuit, do we potentially lose information due to not passing the full flux time series from slave thread back to master. In that case, the estimated relative error (~10^{-13}) is many orders of magnitude smaller than what is of consequence for satellite design.

Implementation of these algorithms will enable the large orbit surveys, long mission simulations, and numerous Monte Carlo scenarios that enable sophisticated decision making by modern satellite designers.

References

Ginet., G. P., T. P. O’Brien, S. L. Huston, W. R. Johnston, T. B. Guild, R. Friedel, C. D. Lindstrom, C. J. Roth, P. Whelan, R. A. Quinn, D. Madden, S. Morley, and Y. J. Su, “AE9, AP9 and SPM: New models for specifying the trapped energetic particle and space plasma environment,” *Space Sci. Rev.,* 2013, doi:10.1007/s11214-013-9964-y.

O’Brien, T. P. Preliminary algorithms for computation of mission aggregate and worst-case fluxes from the empirical AE-9/AP-9 Model,” TOR-2007(3905)-18, The Aerospace Corporation, El Segundo, CA, September 2007.

Seltzer, S. M., “Updated calculations for routine space-shielding radiation dose estimates: SHIELDOSE-2,” Gaithersburg, MD, NIST Publication NISTIR 5477, 1994