



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Design, Implementation and Optimization of a Parallel Monte Carlo Particle Transport Code

R. J. Procassini, J. M. Taylor, G. M. Greenman,
M. J. O'Brien, D. E. Cullen

September 7, 2004

Computational Methods in Transport Workshop
Tahow City, CA, United States
September 11, 2004 through September 16, 2004

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

Design, Implementation and Optimization of a Parallel Monte Carlo Particle Transport Code

Richard Procassini, Janine Taylor, Gregory Greenman,
Matthew O'Brien and Dermott Cullen
The Mercury Code Development Team
Lawrence Livermore National Laboratory



Computational Methods in Transport Workshop
11 – 16 September 2004

University of California



Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94551

Outline



- *The Monte Carlo Method of Particle Transport*
- *Description of the Mercury Monte Carlo Transport Code*
- *The Mercury Parallel Programming Model*
- *Parallel Performance of the Mercury Monte Carlo Code*

The Monte Carlo Method of Particle Transport



- The Monte Carlo method of simulating particle transport is a *statistical* approach to “solving” the linearized Boltzmann equation:

$$\begin{aligned}
 & \frac{1}{v} \frac{\partial \psi(\vec{r}, E, \Omega, t)}{\partial t} + && \text{Temporal Streaming} \\
 & (\nabla \cdot \Omega) \psi(\vec{r}, E, \Omega, t) + && \text{Spatial Streaming} \\
 & \Sigma_a \psi(\vec{r}, E, \Omega, t) = && \text{Collisional Absorption} \\
 & \int_{E'} \int_{\Omega'} \Sigma_s(\vec{r}, E', \Omega' \rightarrow E, \Omega) \psi(\vec{r}, E', \Omega', t) d\Omega' dE' + && \text{Collisional Scattering} \\
 & \chi(E) \int_{E'} v \Sigma_f(\vec{r}, E') \int_{\Omega'} \psi(\vec{r}, E', \Omega', t) d\Omega' dE' + && \text{Collisional Fission Source} \\
 & S_{ext}(\vec{r}, E, \Omega, t) && \text{External Source}
 \end{aligned}$$

- While the spatial domain is divided into cells or regions, and energy may be divided into groups, this method does *not* employ a *continuum* approach to solve this integro-differential equation, as is the case in deterministic transport methods (S_N , P_N , etc)
- The essence of the Monte Carlo method of particle transport is to follow, or *track*, the trajectory of *individual* particles through this eight-dimensional phase space in an *analog* fashion.

The Monte Carlo Method of Particle Transport



- Particles undergo a series of events during tracking:
 - Streaming to the end of the time step: *Census Event* (**Temporal Streaming**)
 - Streaming to the boundary of a neighboring cell, region or system boundary: *Facet Crossing Event* (**Spatial Streaming**)
 - Streaming to the system boundary results in leakage: *Escape Event*
 - Interaction with an atom or nucleus in the background medium: *Collision Event* (**Collisional Absorption**, **Collisional Scattering**)
 - Collisional interactions may result in the production of secondary particles (**Collisional Fission Source**)
 - Streaming to the lower-energy group boundary or thermal energy of the medium during charged-particle slowing down: *Energy-Boundary Crossing* or *Thermalization*
- The trajectory, or track, of each particle through phase space is comprised of several segments.
- Each Monte Carlo simulation particle represents an *ensemble* of physical particles.

The Monte Carlo Method of Particle Transport



Mesh-Based Monte Carlo Particle Tracking

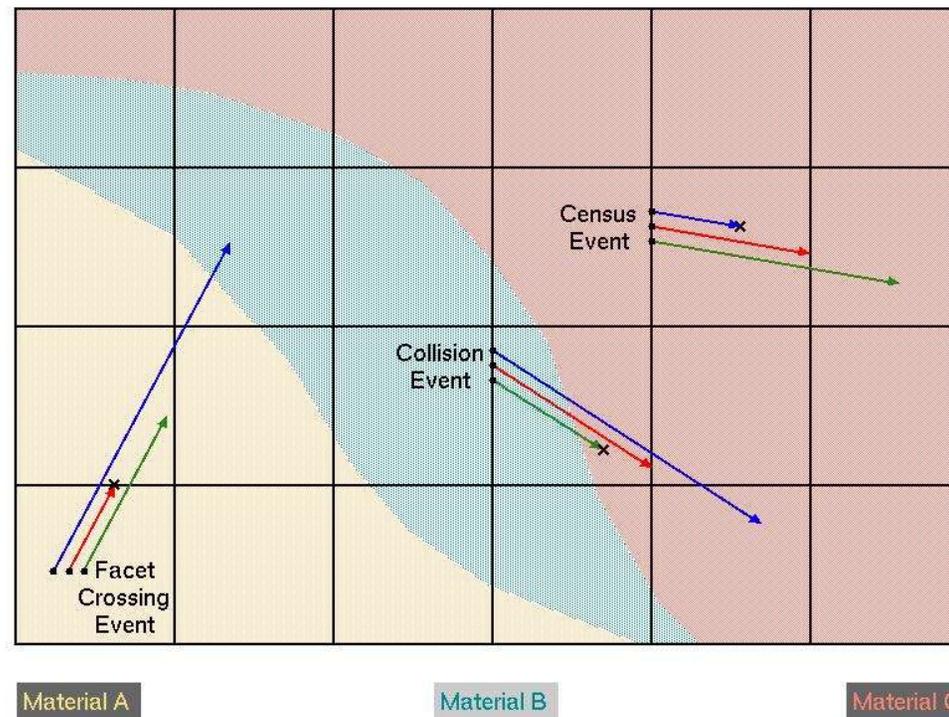
Each particle track is comprised of several segments.

Single segments from the tracks of three particles are shown.

The event chosen for each segment is the one with the shortest distance.

Note: The event distances are actually colinear.

-  Distance to Census
-  Distance to Facet Crossing
-  Distance to Collision



The Monte Carlo Method of Particle Transport



- Tracking of particles in three-dimensional Cartesian space is independent of the problem geometry or mesh: a particle travels in a straight line until it intersects a facet/surface of first order (plane), second order (sphere, cylinder, cone, ellipsoid), etc.
- While the particles have discrete energies, the multigroup treatment of energy employs cross sections which are constant within a group.
- Collisions are point events which may result in the production of secondary particles:
 - Particles which are being tracked are placed in the vault for subsequent tracking
 - Particles which are *not* being tracked are assumed to be locally deposited
- The flux in a given cell (c) and energy group (g) is sum of the particle (i) path lengths (l) through the volume (V) in a given time step (Δt) :

$$\phi_{c,g} = \sum_i \left(\frac{W_i L_{i,g}}{V_c \Delta t} \right)$$

- These fluxes are multiplied by the number densities of background isotopes and the relevant cross sections to produce rates of energy deposition and isotopic burnup.

The Monte Carlo Method of Particle Transport



- Pseudo random numbers are used to randomly sample from various distributions, hence the statistical nature of the method.
- Since the Monte Carlo approach to particle transport is a statistical method, integrated results x will have associated uncertainties which vary as:

$$\frac{\Delta x}{x} = C_x (N_{sim})^{-1/2}$$

where $\Delta x/x$ is the statistical error, is a constant and N_{sim} is the simulation particle count.

- In order to determine the *mean* value for a specific integrated result, x an *ensemble* of simulations is required. This process allows one to determine the value of the constant C_x , which is dependent on the type of problem being simulated.

Description of the Mercury Monte Carlo Transport Code



- The main physics capabilities of Mercury include:
 - Time dependent transport of several types of particles through a medium:
 - Neutrons (n)
 - Gammas (γ)
 - Light charged ions ($^1H, ^2H, ^3H, ^3He, ^4He$)
 - Particle tracking in a wide variety of problem geometries:
 - 1-D radial meshes
 - 2-D r - z meshes
 - 3-D Cartesian and unstructured meshes
 - 3-D combinatorial geometry
 - Multigroup and continuous energy treatment of cross sections
 - Population control can be applied to all types of particles
 - Static k_{eff} and α eigenvalue “settle” calculations for neutrons
 - Dynamic α calculations for all types of particles

Description of the Mercury Monte Carlo Transport Code



- Main capabilities (*continued*):
 - ◆ All types of particles *will* be able to interact with the medium via:
 - Deposition of momentum
 - Deposition of energy
 - Depletion and accretion of isotopes resulting from nuclear reactions
 - ◆ Support for sources is currently limited to:
 - External, mono-energetic or fission spectrum sources
 - External, file-based sources
 - Zonal-based reaction sources
 - ◆ Post-processing tally and diagnostic capabilities are provided by the Caloris code:
 - ◆ Tally capabilities *will* include event history support

Description of the Mercury Monte Carlo Transport Code



- Each Monte Carlo particle is defined by the following attributes:
 - ◆ Spatial coordinates: (x, y, z)
 - ◆ Velocities or Direction Cosines: (v_x, v_y, v_z) or $(\cos(\alpha), \cos(\beta), \cos(\gamma))$
 - ◆ Kinetic Energy: E
 - ◆ Weight: $W = (N_{phys} / N_{sim})$
 - ◆ Time to Census: t_{cens}
 - ◆ Number of Mean-Free Paths to Collision: N_{mfp}
 - ◆ Random Number Seed: R_{seed}
 - ◆ Miscellaneous Attributes: number of collisions, last event, breed, domain, cell, facet, etc.
- These particles are usually tracked in the eight-dimensional phase space comprised of three spatial coordinates (x, y, z) , three velocities (v_x, v_y, v_z) , energy (E) and time (t) .

The Mercury Parallel Programming Model



- The design and development of Mercury has been driven by the following requirements:
 - The code must operate on the wide variety of parallel computing platforms provided by the ASC program
 - The code must be capable of solving problems in one, two and three spatial dimensions, and several types of problem geometries
- To address these requirements, we have adopted a three-pronged approach to parallelism with Mercury:
 - *Domain Decomposition*: Spatial partitioning of the mesh or geometry allows Mercury to track particles through large, multidimensional meshes (*Spatial Parallelism*)
 - *Domain Replication*: Distribution of the particle load across redundant copies of the spatial domain allows Mercury to track a large number of particles (*Particle Parallelism*)
 - *Task Decomposition*: The main particle loop is decomposed by assigning tasks to threads (*Particle Parallelism*)

The Mercury Parallel Programming Model



Domain Decomposition (Spatial Parallelism)

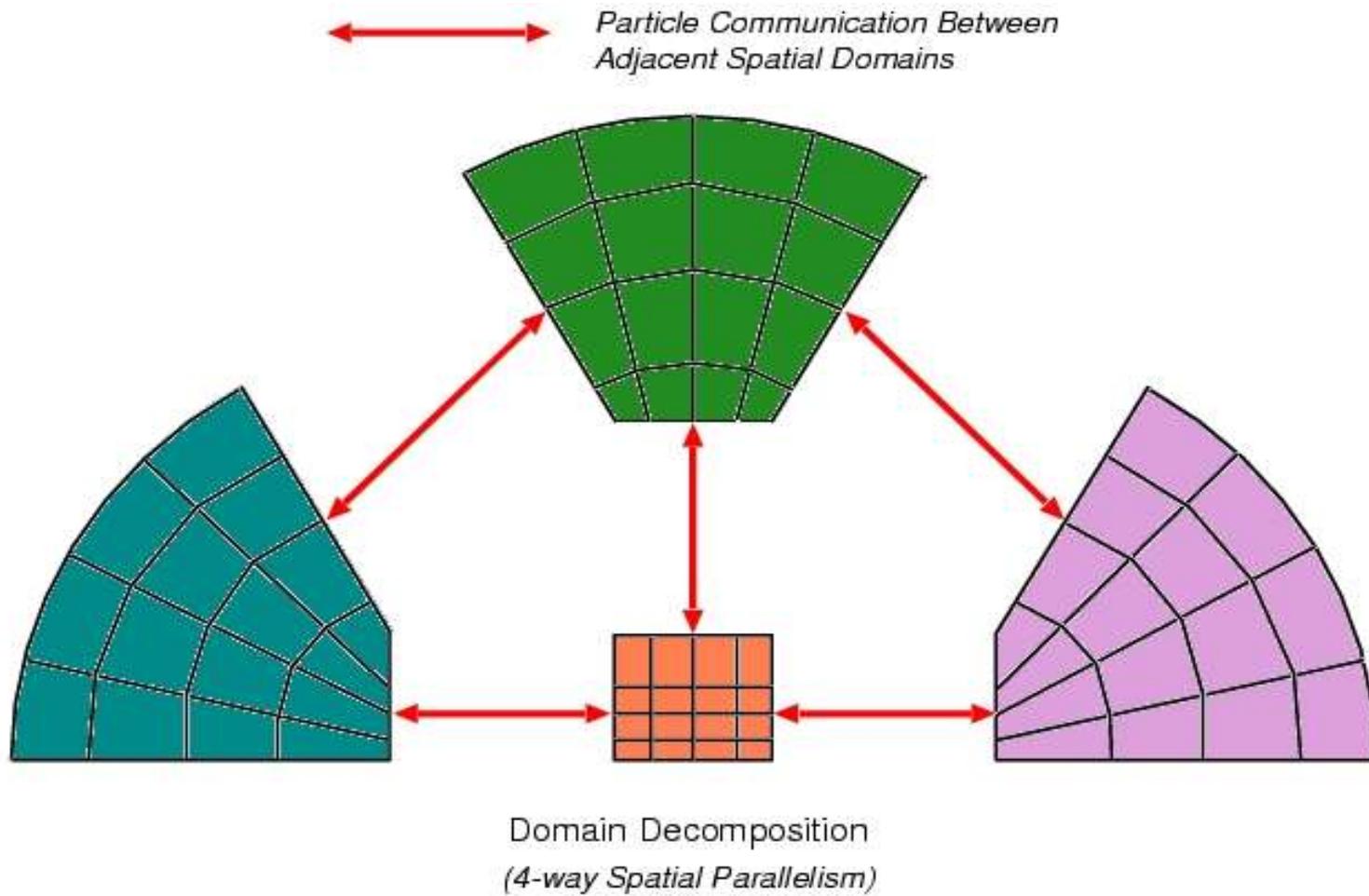
- The non-deterministic nature of Monte Carlo transport *computations*, in the context of a spatially-partitioned geometry/mesh, results in non-deterministic *communication* patterns among adjacent spatial domains.
- Particle that track to a zonal facet which lies on the boundary of a spatial domain must be communicated to the processor containing the adjacent spatial domain.
- The Message Passing Interface (MPI) library provides a portable mechanism for the *point-to-point* communication of particle data between spatial domains:
 - The communication of particle buffers to adjacent domains is currently implemented via the non-block, two-sided communication routine available in MPI-1:

`MPI_Ssend` and `MPI_Irecv`

- Improved parallel performance may be possible via the use of the low-latency, one-sided, remote memory access (RMA) communications routines defined in MPI-2:

`MPI_Put` and `MPI_Get`

The Mercury Parallel Programming Model



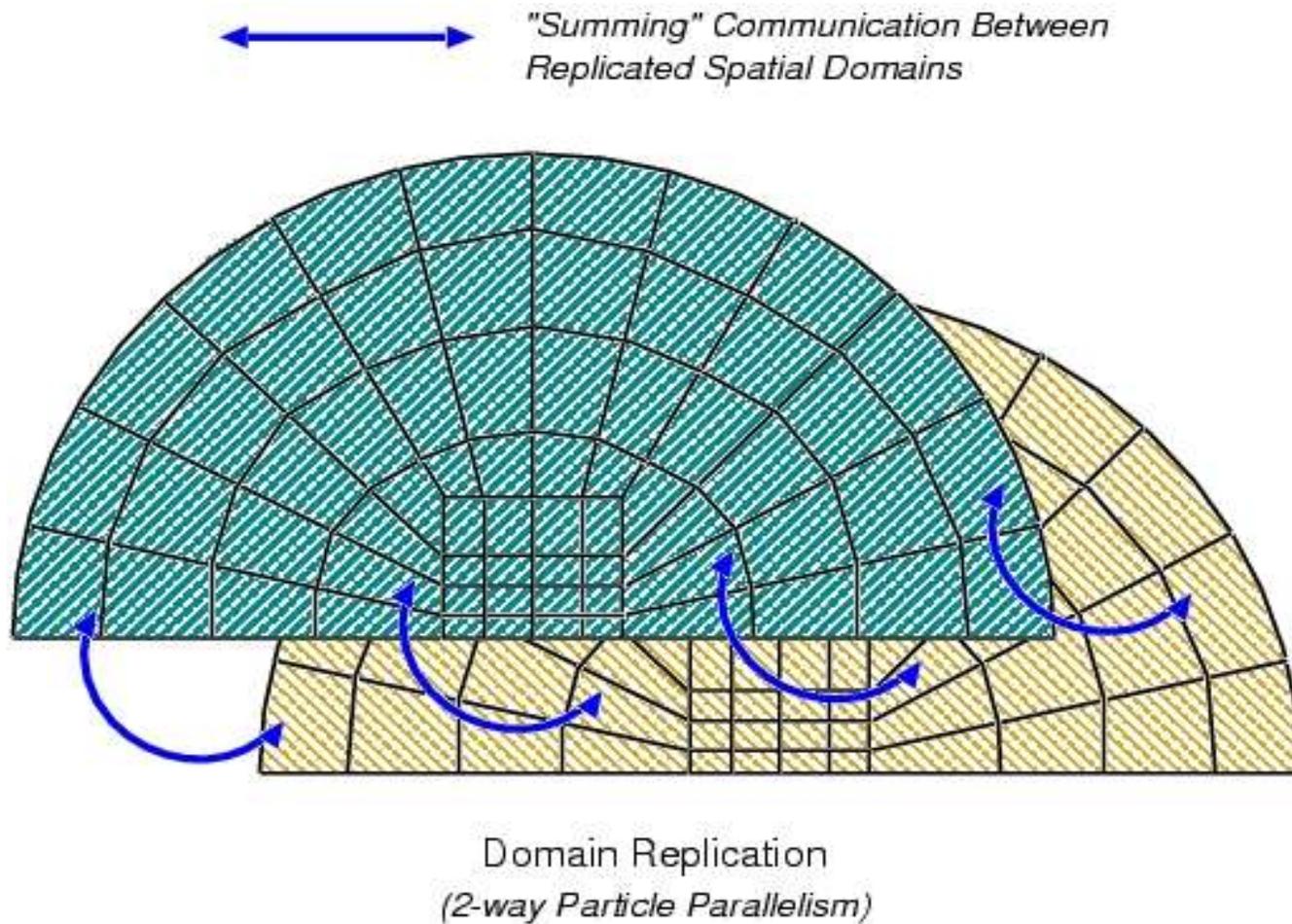
The Mercury Parallel Programming Model



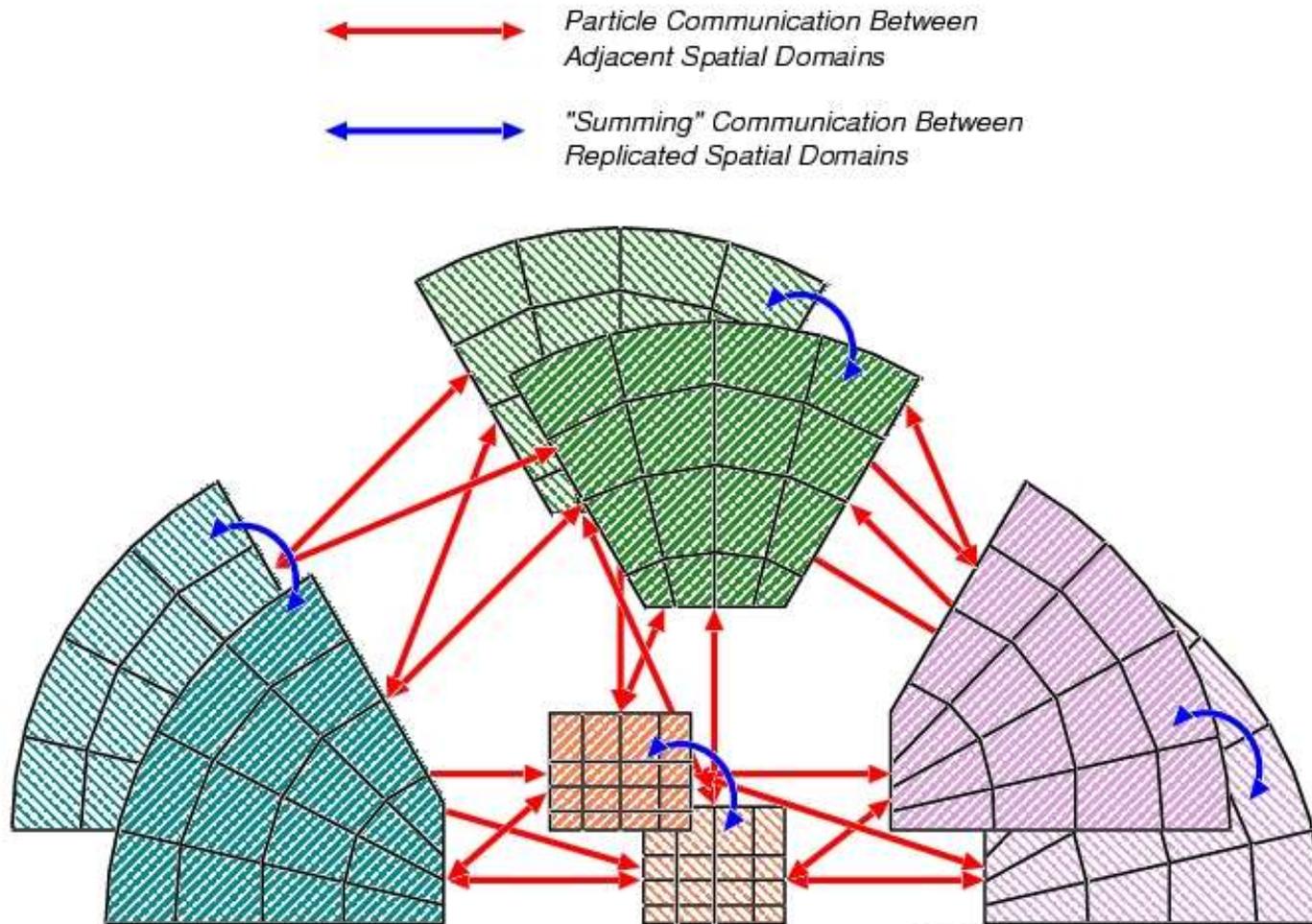
Domain Replication (Particle Parallelism)

- The distribution of the particle load across replicated spatial domains results in the need for tally data to be *summed* across the associated particle work group, including:
 - Particle populations used in k_{eff} and α eigenvalue calculations
 - Scalar fluxes
 - Isotopic mass depletion/production in the medium
 - Energy deposition to the medium
- The MPI library is also used to implement the *collective* communication of data among replicated domains:
 - The summing operations are handled via *reduction* operations among the members of a replicated work group: `MPI_Reduce` and `MPI_Bcast`
 - The performance of these reduction operations has been observed to degrade as the size of the work group increases

The Mercury Parallel Programming Model



The Mercury Parallel Programming Model



Domain Decomposition and Domain Replication
(4-way Spatial and 2-way Particle Parallelism)

The Mercury Parallel Programming Model



Task Decomposition (Particle Parallelism)

- The main particle loop is parallelized using a thread-based task model, where groups of particles are assigned to each thread.
- To avoid the expense of atomically-locking shared data structures during *summing* operations, task replicated (redundant) versions of all writable data structures are created.
- The summing operation is thereby reduced to the addition of these replicated (non-shared) data structures.
- The Open-MP library is used to implement task-based threading within the shared-memory programming model:
 - A threaded section that encompasses the main particle loop: `pragma omp parallel`
 - Critical sections within the thread-parallel section that protect the (apparently) non-thread-safe MPI point-to-point communications: `pragma omp critical`

The Mercury Parallel Programming Model



- The use of a message passing paradigm for the implementation of Domain Decomposition and Domain Replication has produced the following processor hierarchy:
 - A *Worker* processor transports particles on one of the replicated spatial domains (Domain Replication)
 - A *Foreman* processor performs all the tasks of a *Worker* processor, and also controls the operations of the *Worker* processors in its work group (Domain Decomposition and Domain Replication). The additional duties of a *Foreman* processor include:
 - Parallel I/O to/from restart dumps and to graphics dumps
 - A *Boss* processor performs all the tasks of a *Foreman* processor, and also controls the overall operation of the code (Domain Decomposition and Domain Replication). The additional duties of a *Boss* processor include:
 - Serial I/O from input files and to edit files
 - Calculation of all critical (serial) sections of the code

The Mercury Parallel Programming Model



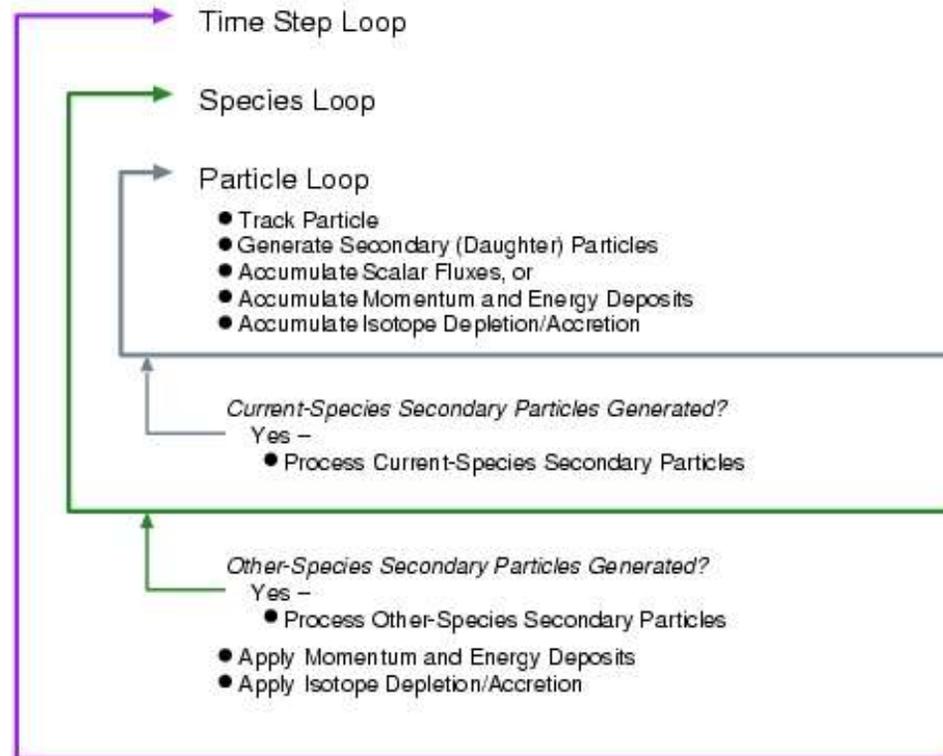
Serial Monte Carlo Particle Transport

- The flow control of a *serial*, time-dependent, multi-species Monte Carlo particle transport code can be implemented as a set of nested `while` loops:
 - Loop (3) over the time steps, surrounding
 - Loop (2) over the particle species, surrounding
 - Loop (1) over the particles of a specific species
- Secondary (daughter) particles that are generated during a pass through Loop (1) are processed in later passes.
- If necessary, multiple passes are made through Loop (1) until *all* particles of the *current* species are tracked to *census* at the end of the time step.
- If necessary, multiple passes are made through Loop (2) until *all* particles of *all* species are tracked to census.

The Mercury Parallel Programming Model



Serial Monte Carlo Nested Loop Structure



The Mercury Parallel Programming Model



Parallel Monte Carlo Transport

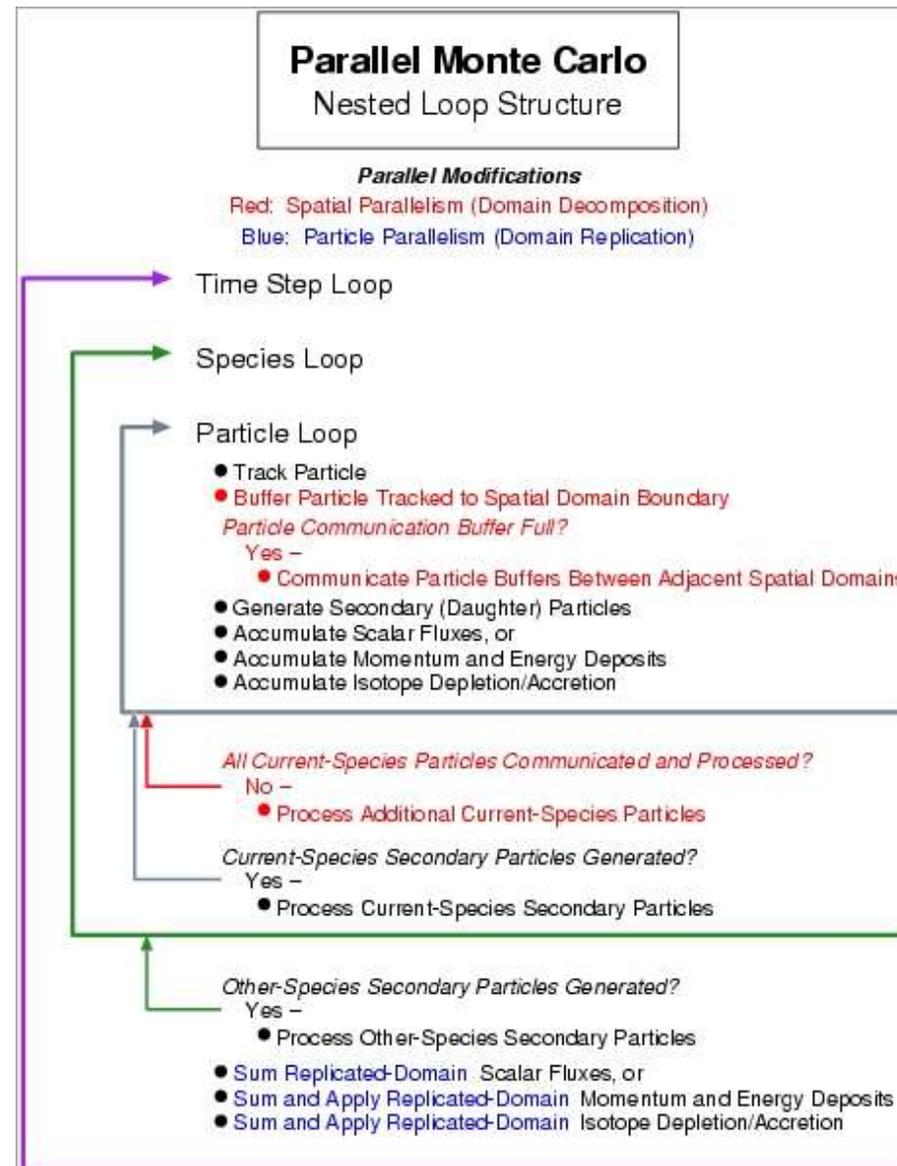
- The non-deterministic communication patterns that result from spatial partitioning of the computational geometry must be dealt with in an optimal manner to avoid large degradations of parallel efficiency.
- In an effort to reduce the time required to communicate particles between adjacent spatial domains:
 - Particles are buffered, according to destination domain, when they track to a spatial domain boundary within Loop (1) for later communication
 - In an effort to overcome communication latencies, buffers of particles are communicated rather than individual particles
 - The communication buffer is sent to the appropriate, adjacent spatial domain when it (a) becomes full during Loop (1) or (b) the current pass through Loop (1) ends
 - The communication of partially-filled buffers at the end of Loop (1) can reduce the efficiency of the parallel algorithm

The Mercury Parallel Programming Model



- Additional interprocessor communication is required for the calculation of tallies, including:
 - Summing the particle population over *all* processors prior to k_{eff} or α eigenvalue calculations
 - Summing the scalar flux over *domain replicated* processors
 - Summing the energy deposits over *domain replicated* processors
 - Summing isotopic depletion and production over *domain replicated* processors

The Mercury Parallel Programming Model



Parallel Performance of the Mercury Monte Carlo Code



- A series of 2-D and 3-D calculations were performed in order to assess the parallel efficiency of Mercury.
- The efficiency of both the domain decomposition *and* domain replication algorithms were studied.
- These calculations were performed on the ASC White parallel computer at LLNL: AN IBM SP-2 system with 16-way SMP nodes.
- The two criticality problems chosen for this study are:
 - ♦ *Planet (HEU-MET-FAST-018)*: A k_{eff} calculation of a beryllium-reflected, plutonium critical assembly
 - ♦ *Double-Density Jezebel*: A settle α calculation of the *gedanken* double-density Jezebel super-critical device

Parallel Performance of the Mercury Monte Carlo Code



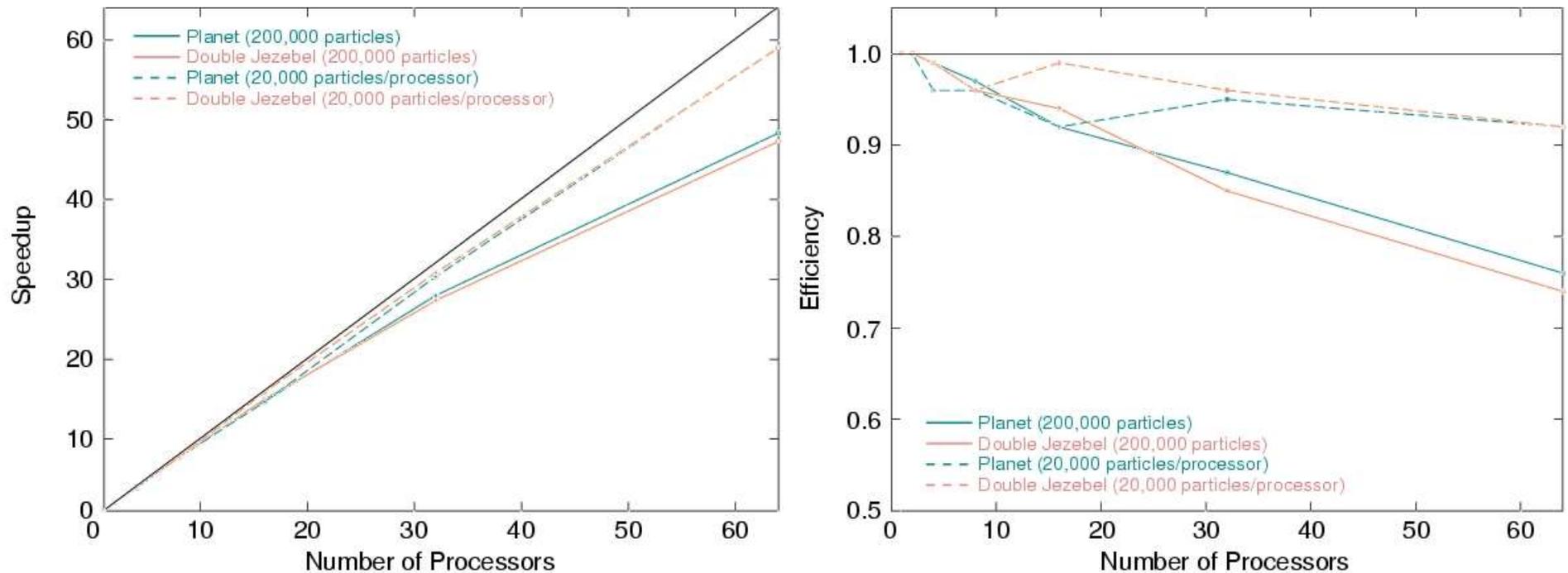
- These calculations (2-D / 3-D) were run with:
 - ♦ *Constant Work Load*: A constant particle count of $N_{sim} = 200,000$ (2-D) or $N_{sim} = 1,000,000$ (3-D) as the processor count was varied in the range $2 < N_{proc} < 64$ (2-D) or $2 < N_{proc} < 512$ (3-D)
 - ♦ *Constant Work Load per Processor*: A constant particle count of $N'_{sim} = 20,000$ (2-D) or $N'_{sim} = 25,000$ (3-D) per processor as the processor count was varied in the range $2 < N_{proc} < 64$ (2-D) or $2 < N_{proc} < 512$ (3-D)

Parallel Performance of the Mercury Monte Carlo Code



Domain Replication

2-D Simulations

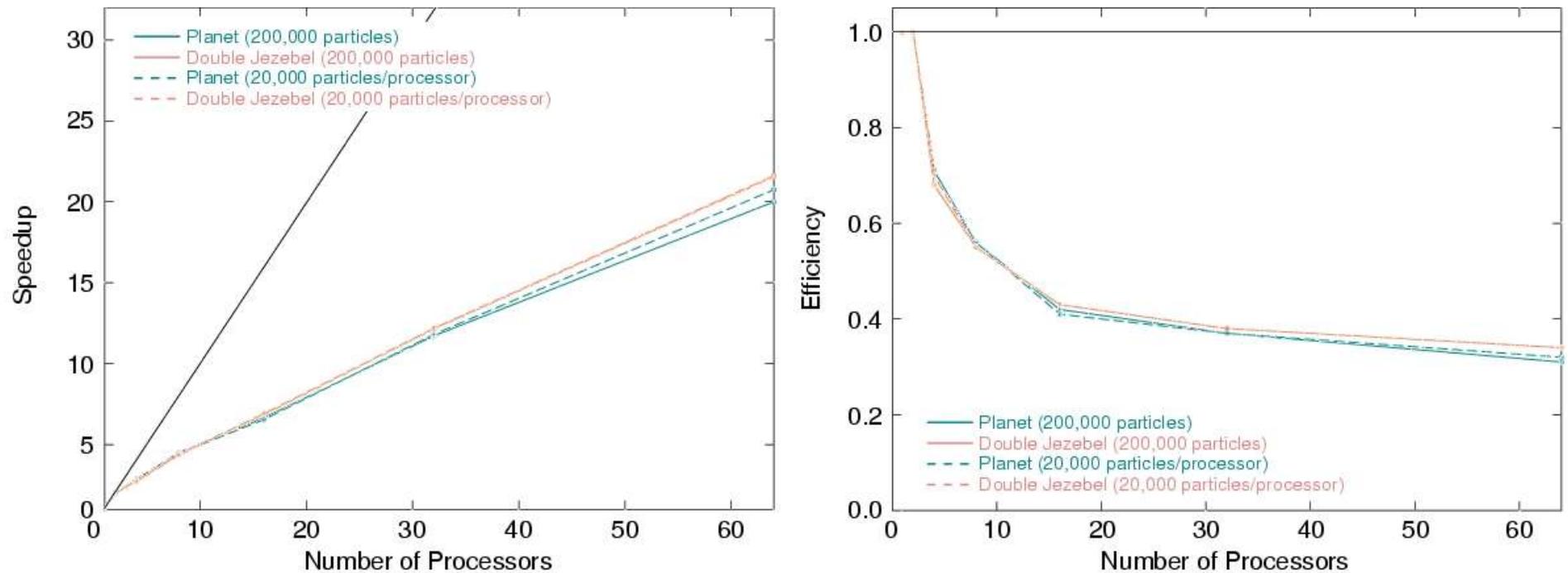


Parallel Performance of the Mercury Monte Carlo Code



Domain Decomposition

2-D Simulations



Parallel Performance of the Mercury Monte Carlo Code



- The capability to *statically* or *dynamically* load balance a calculation has been tested. These methods vary the replication level (\mathbb{R}) of each work group in an attempt to balance the load across all work groups.
- Static load balancing was tested on 2-D calculations of the Planet critical assembly:
 - The total processor count was fixed at $N_{proc} = 64$
 - The replication level (\mathbb{R}) of any work groups was *preassigned* for each simulation in the range $1 < \mathbb{R} < 7$ for a fixed number of spatial domains $N_{dom} = 16$
- Dynamic load balancing was tested on 2-D calculations of the double density Jezebel super-critical assembly:
 - The replication level (\mathbb{R}) of each spatial domain is dynamically changed in an effort to maximize the parallel efficiency by assigning processors to the most worked domains, according to a predefined performance model
 - This method requires that particles periodically be sent between processors as the size of the work groups changes to balance the particle work load per domain

Parallel Performance of the Mercury Monte Carlo Code



Static Load Balancing

Domain Decomposition and Domain Replication

2-D Simulations: Planet Critical Assembly

<i>Minimum Replication Level</i>	<i>Maximum Replication Level</i>	<i>Average Replication Level</i>	<i>Run Time [sec]</i>
4	4	4.00	396.26
2	5	4.50	557.17
2	5	4.00	562.38
1	6	4.00	1058.77
1	6	4.00	1062.69
1	6	3.75	553.61
1	7	4.00	1049.88

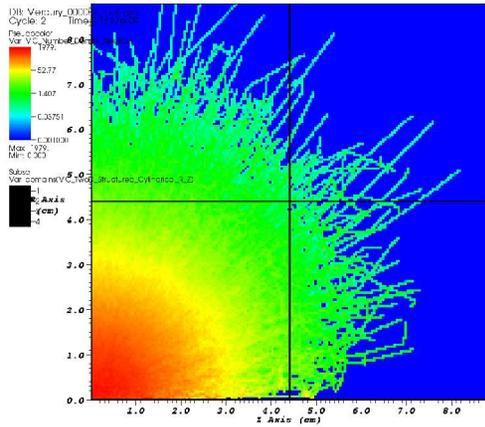
Parallel Performance of the Mercury Monte Carlo Code



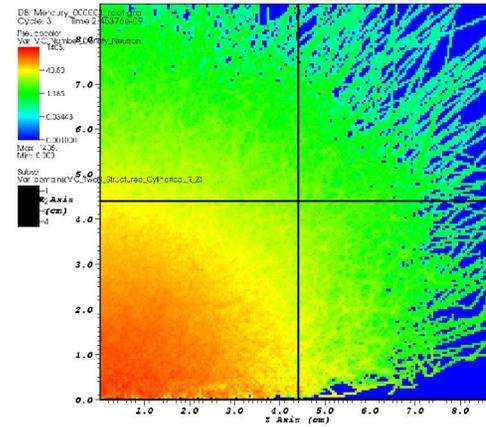
Dynamic Load Imbalance

Pseudocolor plot of particle number density at various times during the simulation.

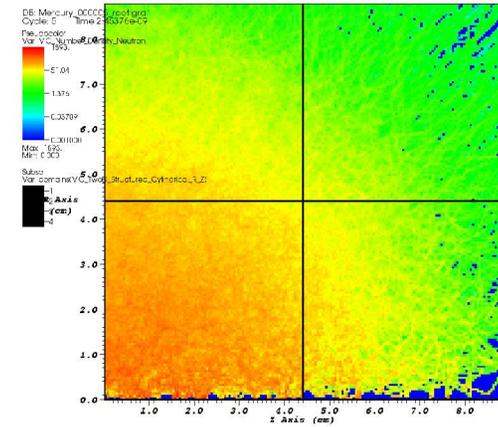
Redder areas represent larger amounts of computational work.



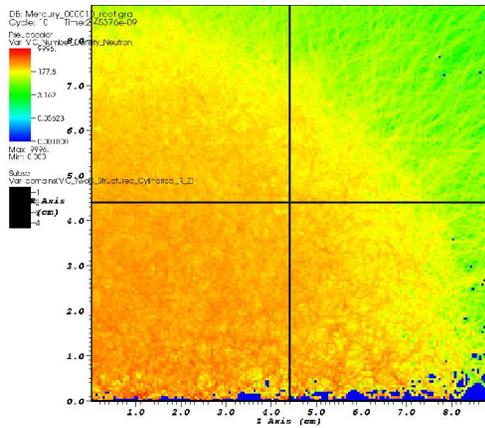
user: mcloran
Tue Aug 31 17:31:53 2004



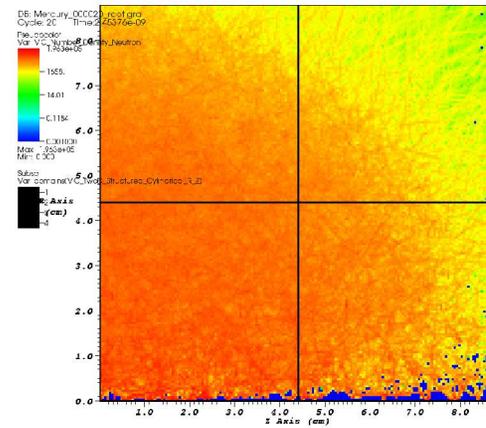
user: mcloran
Tue Aug 31 17:32:37 2004



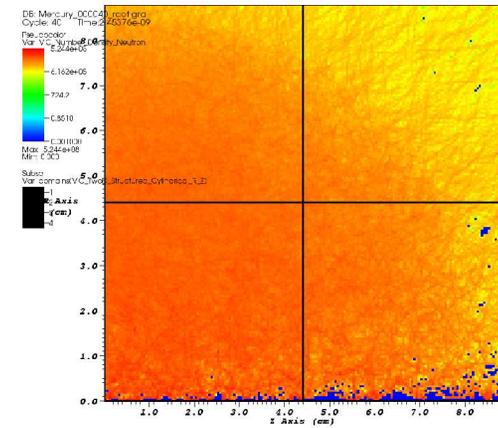
user: mcloran
Tue Aug 31 17:32:59 2004



user: mcloran
Tue Aug 31 17:33:02 2004



user: mcloran
Tue Aug 31 17:33:20 2004



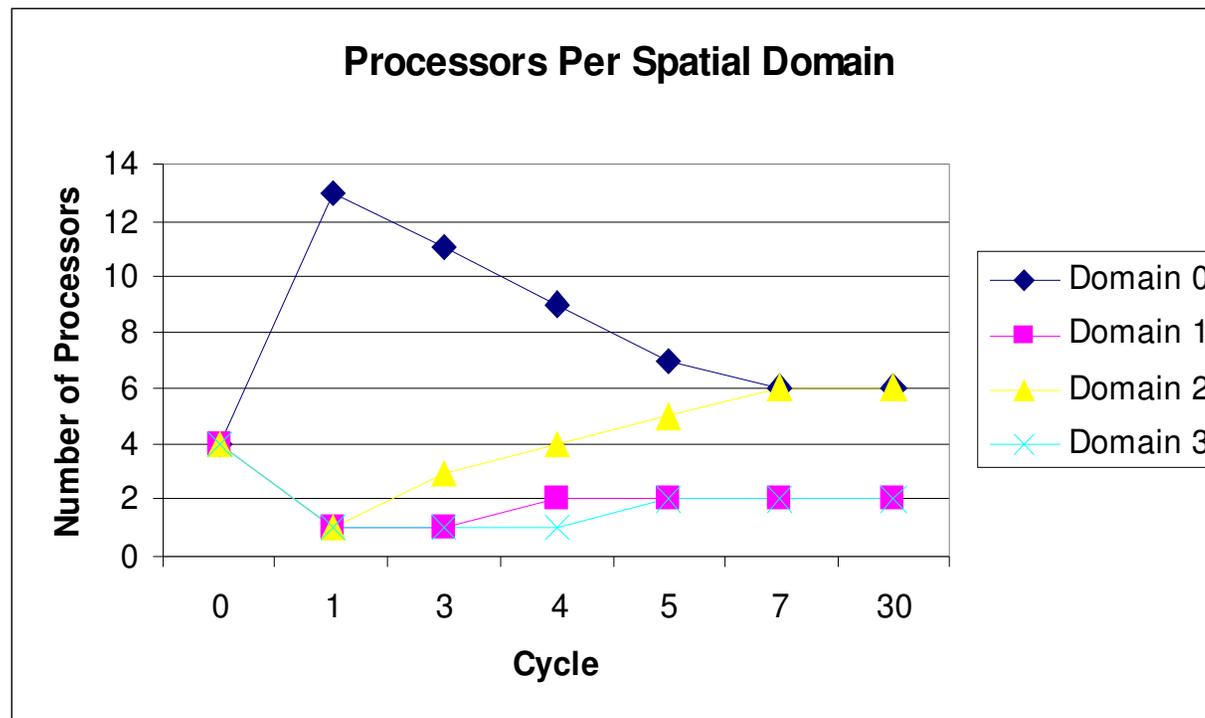
user: mcloran
Tue Aug 31 17:33:32 2004

Parallel Performance of the Mercury Monte Carlo Code

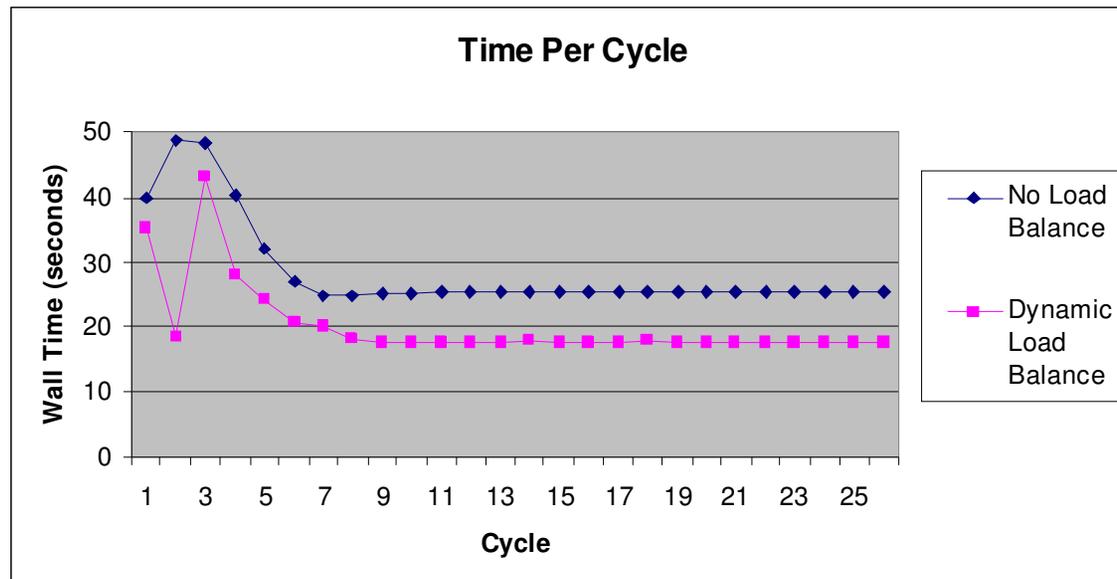
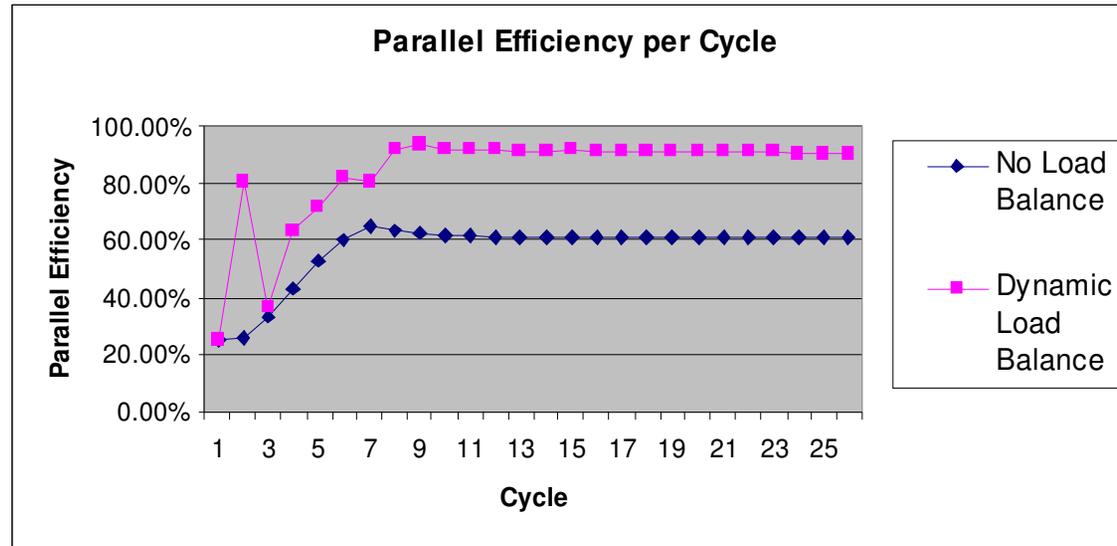


Dynamic Load Balancing

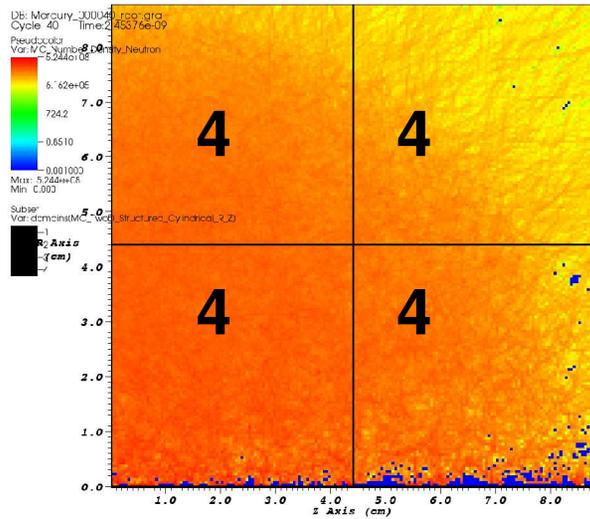
- Initially 4 processors are assigned to each domain.
- Domain 0 is found to have the largest amount of computational work (particle track segments)
- The size of the work groups is varied, particles are communicated and the workload balances out over time. After cycle 7, there are no more changes in the workload.



Parallel Performance of the Mercury Monte Carlo Code

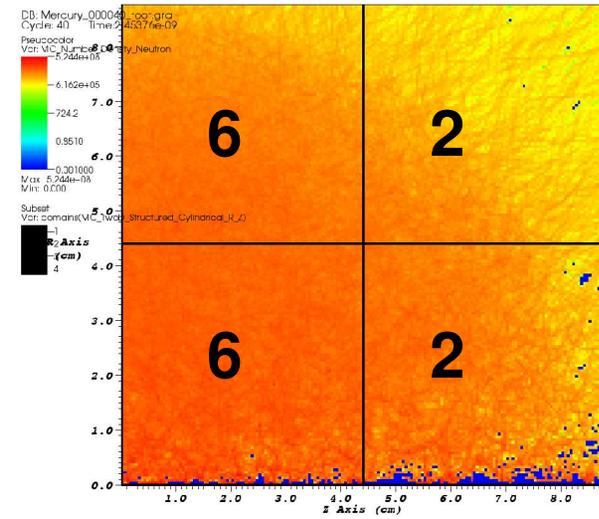


Parallel Performance of the Mercury Monte Carlo Code



user: mobrien
Tue Aug 31 17:33:32 2004

Uniform assignment of processors
to domains: 60% efficient.



user: mobrien
Tue Aug 31 17:33:32 2004

Assign processors to domains based
on work per domain: 91% efficient.

- Processors “diffuse” to the most worked domains.
- The performance model assumes that the computational work is the number of *segments*: cell facet crossings, collisions, censuses, etc. This model has a 97% correlation with actual wall time.
- Parallel efficiency is defined as t_{ave} / t_{max} or (ave time over all processors) / (max time over all processors).