Dynamic Load Balancing of Parallel Monte Carlo Transport Calculations

Richard Procassini, Matthew O'Brien and Janine Taylor
Lawrence Livermore National Laboratory

Joint Russian-American Five-Laboratory Conference
on Computational Mathematics and Physics
19 – 23 June 2005
Vienna, Austria

University of California
Lawrence Livermore National Laboratory

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

Outline

- *Description of the MERCURY Monte Carlo Code*
- *The MERCURY Parallel Programming Model*
- *Load Imbalance in Parallel Monte Carlo Calculations*
- *The MERCURY Load Balancing Algorithm*
- *The Results of Dynamic Load Balancing in MERCURY*
- *Summary and Conclusions*

Description of the MERCURY Monte Carlo Code

- The main physics capabilities of the MERCURY Monte Carlo transport code include:
  - Time dependent transport of several types of particles through a medium:
    - Neutrons (n)
    - Gammas (γ)
    - Light charged ions $({}^{1}H, {}^{2}H, {}^{3}H, {}^{3}He, {}^{4}He)$
  - Particle tracking through a wide variety of problem geometries:
    - 1-D spherical (radial) meshes
    - 2-D *r-z* structured and quadrilateral unstructured meshes
    - 3-D Cartesian structured and tetrahedral 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 Code

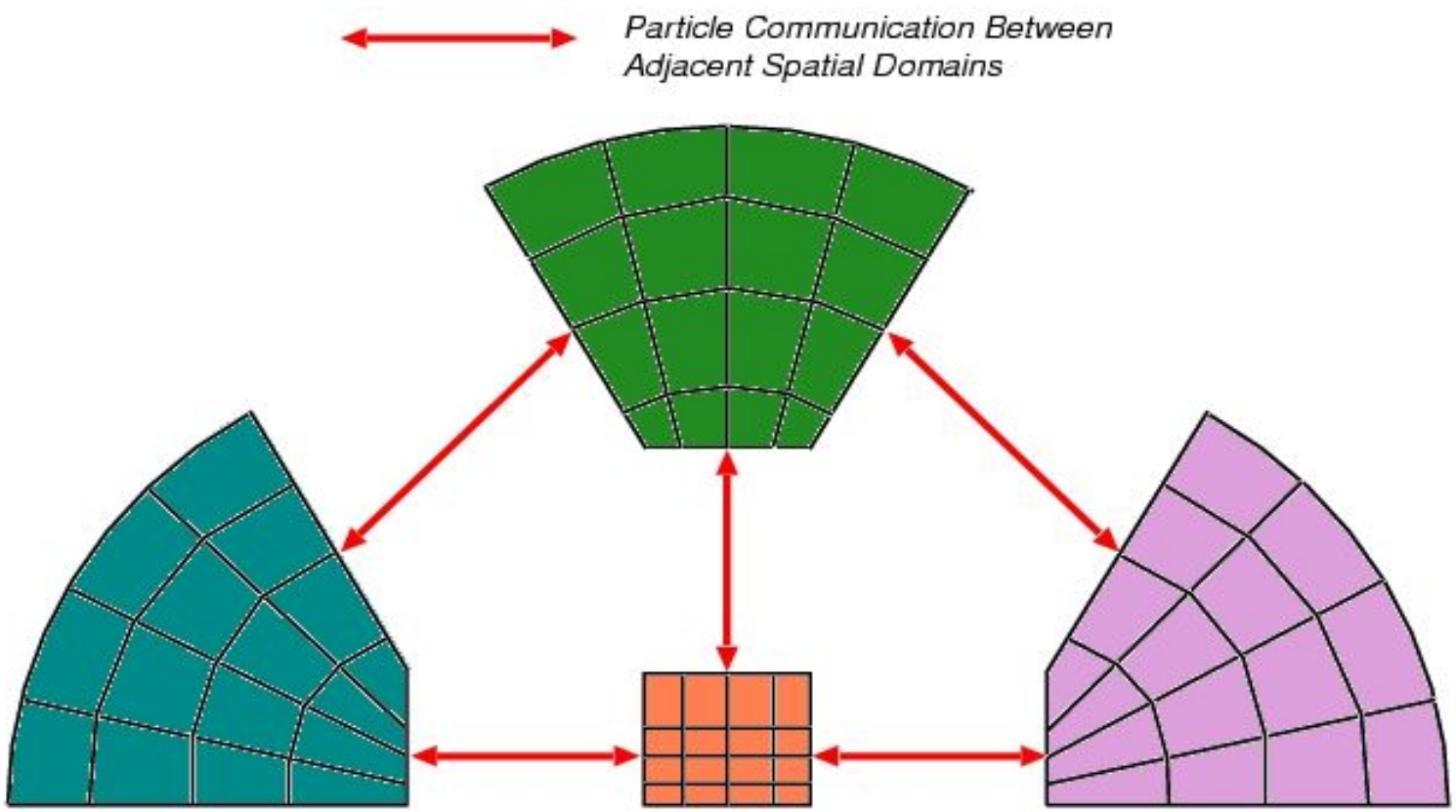
- Main physics capabilities of MERCURY (*continued*):
  - All types of particles can interact with the medium via collisions, resulting in:
    - Deposition of energy
    - Depletion and accretion of isotopes resulting from nuclear reactions
    - Deposition of momentum *(to be added)*
  - Support for sources is currently limited to:
    - External monoenergetic, fission-spectrum or file-based sources
    - Zonal-based reaction sources
- Near term enhancements of MERCURY will include:
  - Generalization of the current source capabilities
  - Generalization of the current tally capabilities, and addition of event history support
  - Post-processing of tallies will be provided by the CALORIS code
  - Addition of several variance reduction methods

The MERCURY Parallel Programming Model

- The MERCURY Monte Carlo particle transport code employs a hybrid approach to parallelism:
  - *Domain Decomposition:* The problem geometry or mesh is spatially partitioned into domains. Individual processors are then assigned to work on specific domains. This form of s*patial parallelism* allows MERCURY to track particles through large, multidimensional meshes/geometries.
  - *Domain Replication:* The easiest way to parallelize a Monte Carlo transport code is to store the geometry information redundantly on each of the processors. Individual processors are then assigned to work on a different set of particles. This form of p*article parallelism* allows MERCURY to track a large number of particles. The number of processors assigned to work on a domain is known as the *replication level* of the domain.
  - Many problems are so large that particle parallelism alone is not sufficient. For these cases, a combination of both spatial *and* particle parallelism is employed to achieve a scalable parallel solution.

The MERCURY Parallel Programming Model

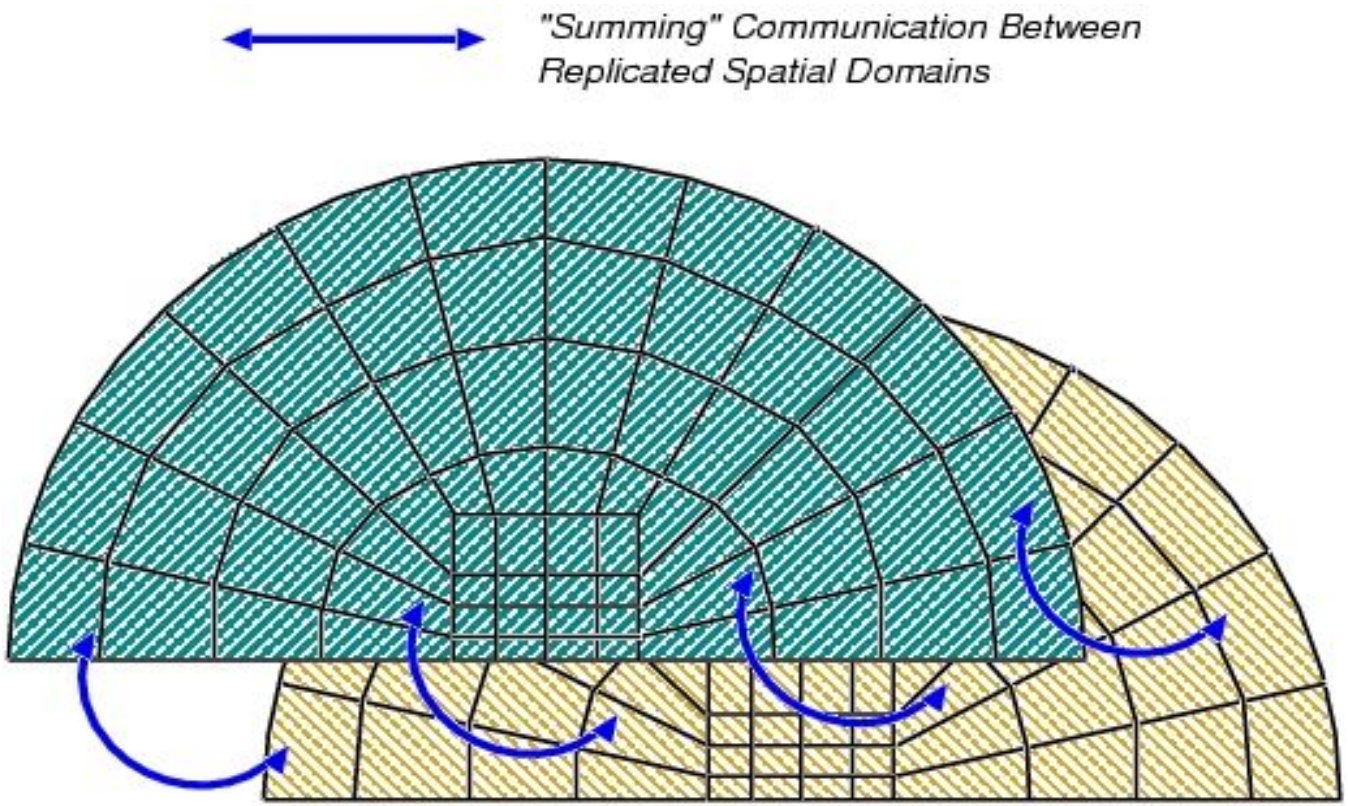
Domain Decomposition *(Spatial Parallelism)*



Domain Decomposition
(4-way Spatial Parallelism)

The MERCURY Parallel Programming Model

Domain Replication *(Particle Parallelism)*

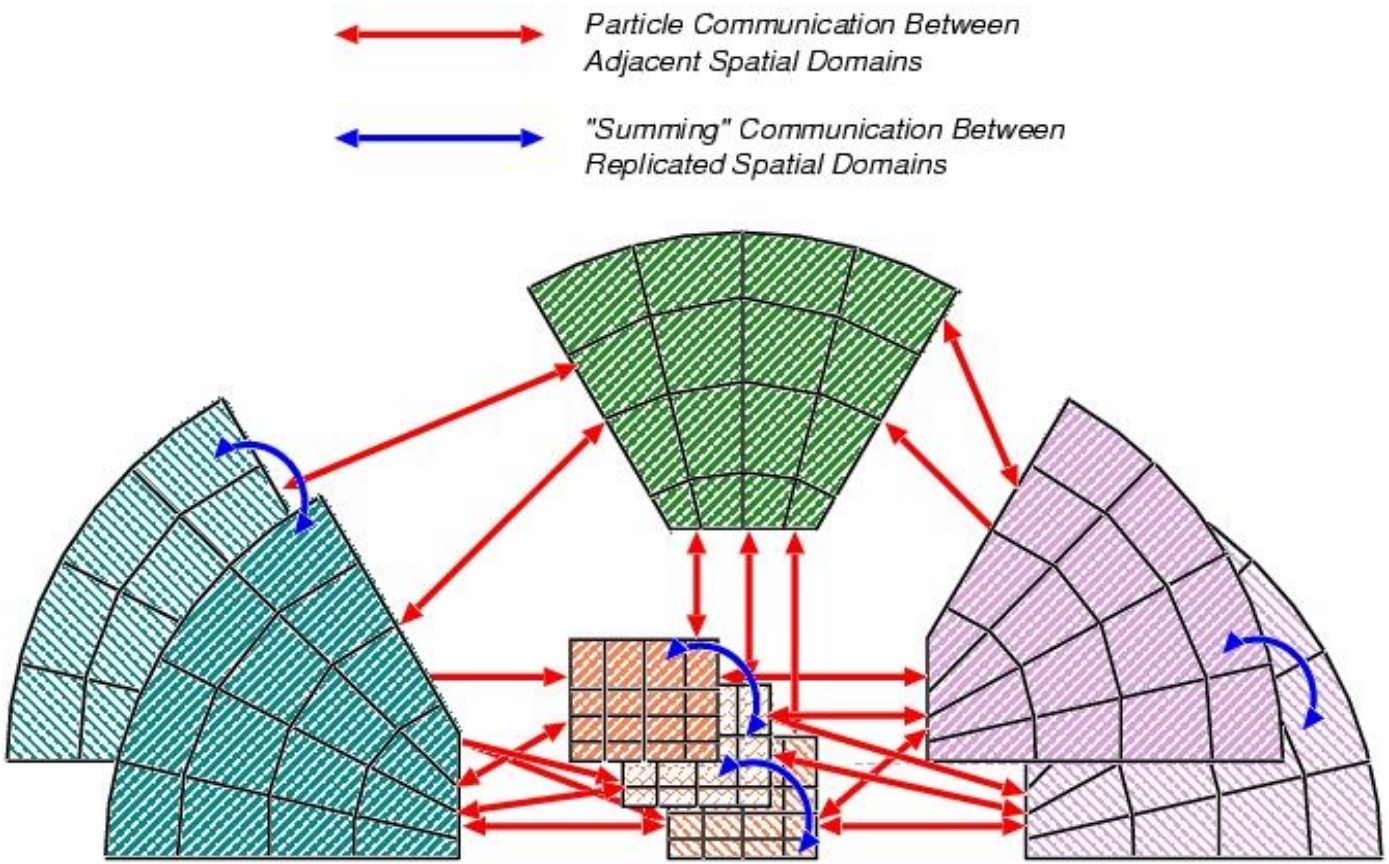


Domain Replication
(2-way Particle Parallelism)

The MERCURY Parallel Programming Model

Domain Decomposition and Domain Replication

(Spatial and Particle Parallelism)



Domain Decomposition and Domain Replication

(4-way Spatial and Variable-way Particle Parallelism)

Load Imbalance in Parallel Monte Carlo Calculations

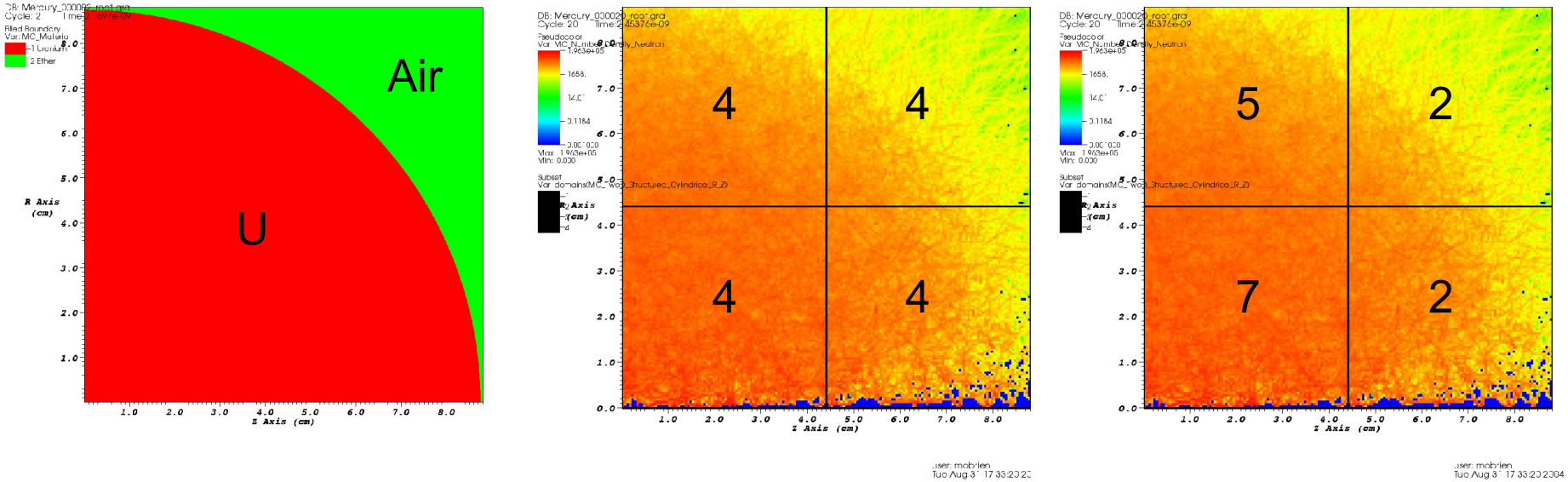
- An important feature of Monte Carlo transport codes is that particles migrate in both space and time between different regions of a problem in response to the physics.
- A natural consequence of domain decomposition is that the amount of computational work will vary from domain to domain.
- Many Monte Carlo algorithms require that one phase of the calculation (cycle, iteration, etc.) must be completed by all processors before the next phase can commence.
- If one processor has more work than the other processors, the less-loaded processors *must* wait for the most-loaded processor to complete its work before proceeding.
- The result is *particle-induced load imbalance*.
- Particle-induced load imbalance can dramatically reduce the parallel efficiency of the calculation, where the efficiency is defined as:

$$\varepsilon = \frac{\overline{W}(p)}{\widehat{W}(p)} = \frac{\left(\frac{1}{N_p}\right)\sum_{p=1}^{N_p}\left[W(p)\right]}{max_{p=1}^{N_p}\left[W(p)\right]}$$

Load Imbalance in Parallel Monte Carlo Calculations

Particle-Induced Load Imbalance

Double-Density Godiva Static α Calculation



(a) Problem Geometry

(b) Uniform: 60% Efficient

(c) Variable: 91% Efficient

4-way Spatial Parallelism (Black lines represent domain boundaries).

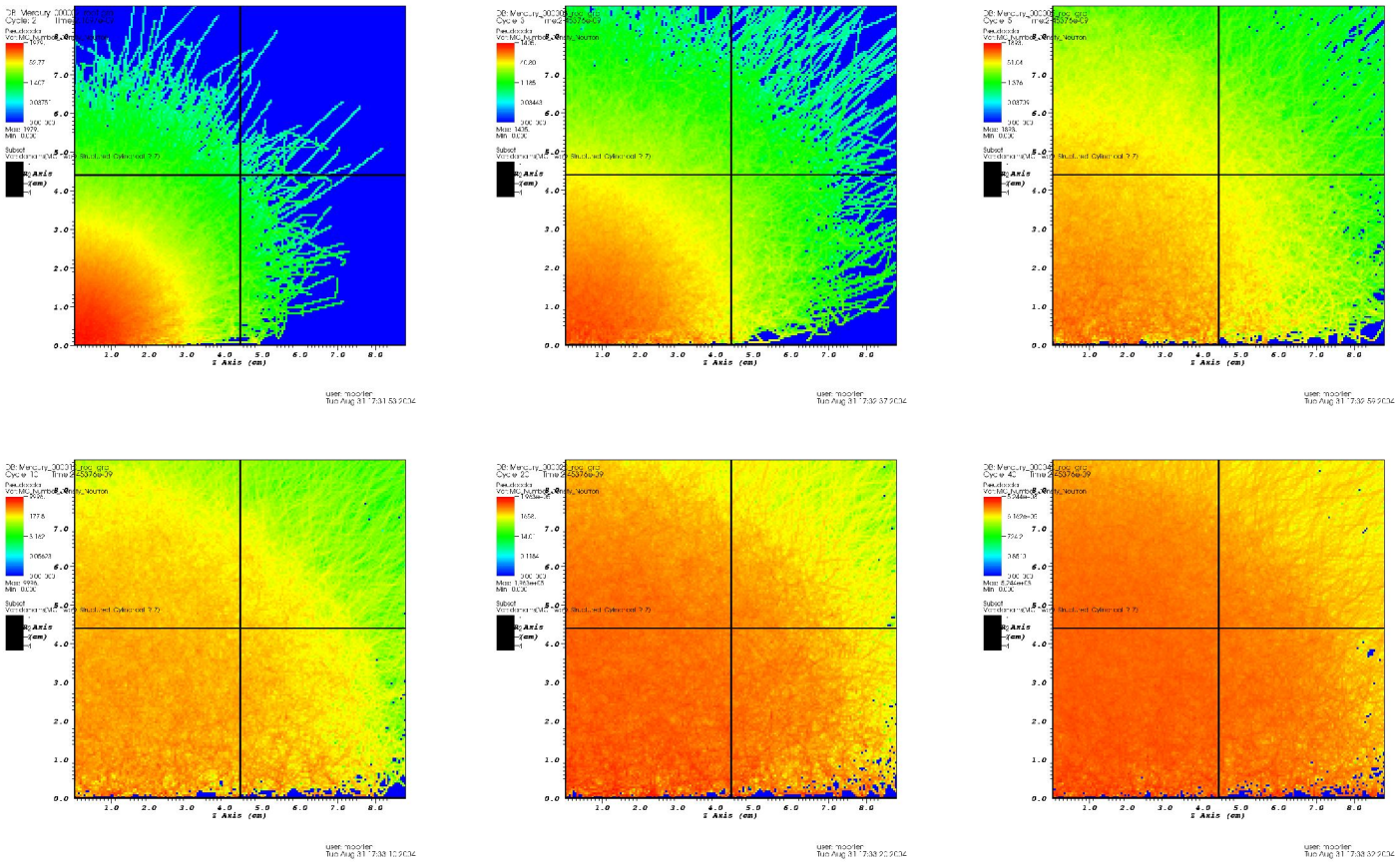
Static Replication of 4 domains on 16 processors (Domain replication level is indicated).

Pseudocolor plots represent neutron number density per cell.

Load Imbalance in Parallel Monte Carlo Calculations

Particle-Induced Load Imbalance

Double-Density Godiva Static α Calculation



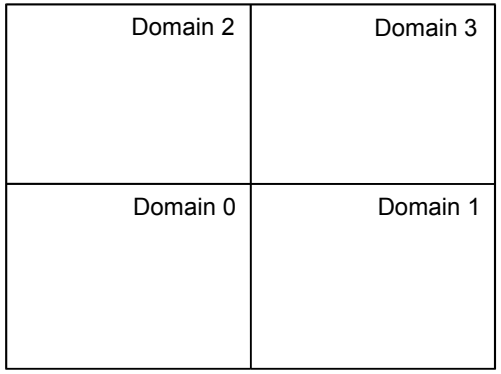
Pseudocolor plots of neutron number density per cell at six cycles during the calculation.

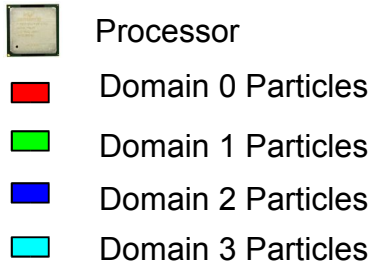
4-way Spatial Parallelism (Black lines represent domain boundaries).

The MERCURY Load Balancing Algorithm

- In an attempt to reduce the effects of particle-induced load imbalance, a load balancing algorithm has been developed and implemented in MERCURY.
- This method is applicable for parallel calculations employing *both* domain decomposition *and* domain replication.
- The essence of the method is that the domain replication level is *dynamically* varied in accordance with the amount of work (particle segments) on that domain.
- When the replication level of a domain is changed, the number of particles located in that domain are distributed evenly among the number of processors assigned to work on the domain. Only the *minimum* number of particles required to achieve a particle-balanced domain are communicated.
- The method periodically checks to determine if the load imbalance is severe enough to warrant the expense of changing the replication levels of the domain, including the cost of communicating particles between processors.

The MERCURY Load Balancing Algorithm

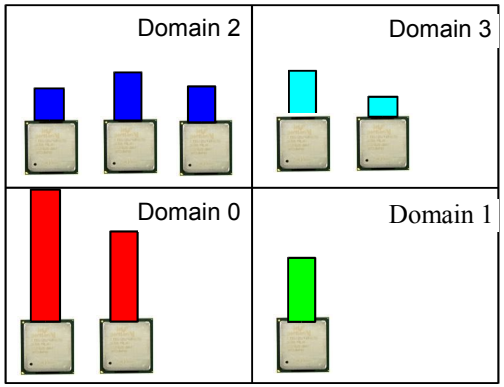




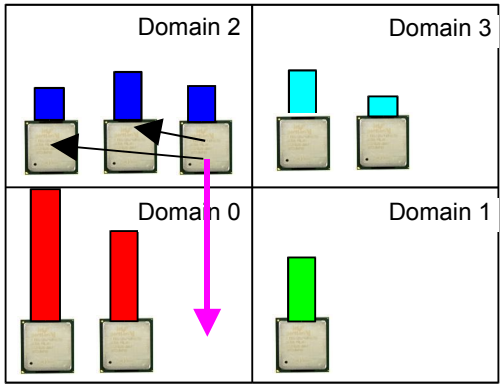
(*a*)

This is the legend for the diagrams in this figure. The length of the particle bar indicates the number of particles on each processor. Particles within a domain must remain within that domain after load balancing.

(*b*)

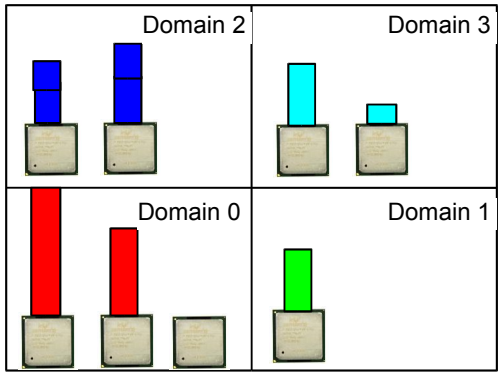


Step 1. The initial particle distribution over processors at the start of a cycle.

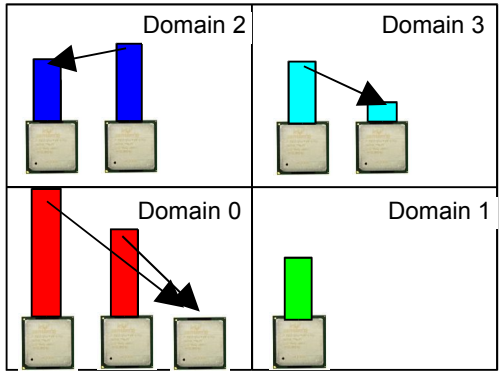


Step 2. Domain 2 loses one processor that is reassigned to Domain 0. The particles on that processor must be communicated to the other processors that remain assigned to Domain 2.

The MERCURY Load Balancing Algorithm



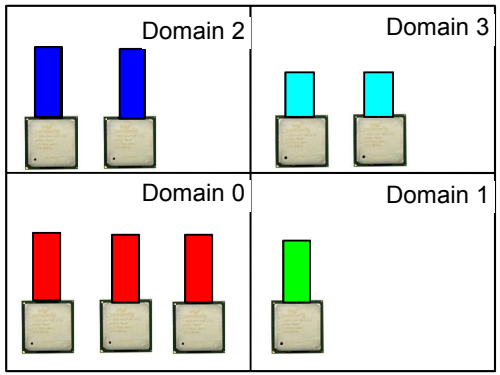
Step 3. After determining which processors are assigned to each domain, each domain can independently balance its particle load.



Step 4. This communication is necessary to achieve load balance within each domain.

(c)

(d)

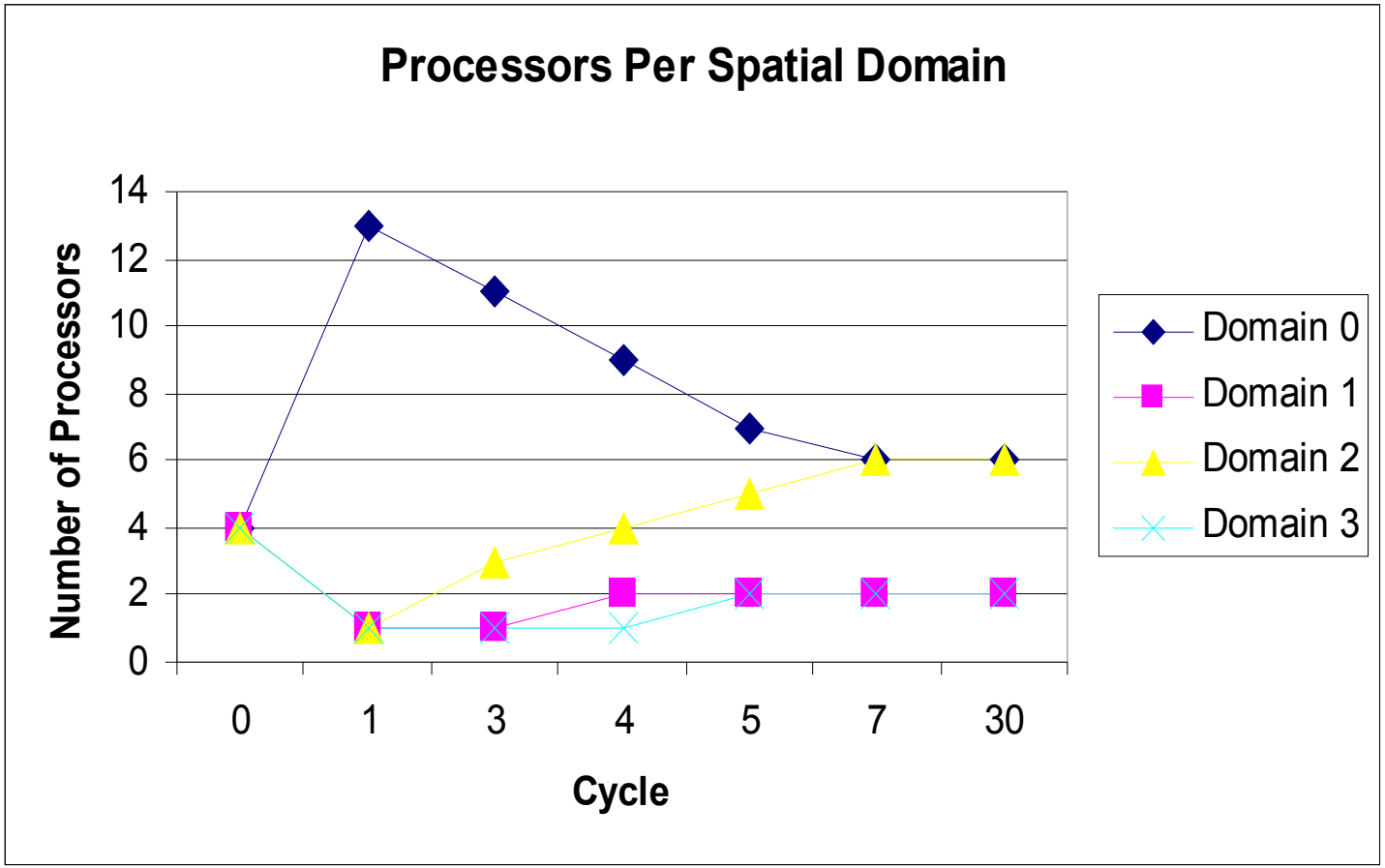


Step 5. The end result of load balancing: the number of processors per domain has been changed so that the maximum number of particles per processor in minimized

The MERCURY Load Balancing Algorithm

Dynamic, Variable Domain Replication

Double-Density Godiva Static α Calculation



Dynamic Replication of 4 domains on 16 processors.

The Results of Dynamic Load Balancing in MERCURY

- The efficacy of dynamic load balancing in the context of parallel Monte Carlo particle transport calculations is tested by running two test problems: one criticality problem and one sourced problem.
- These problems are chosen because they exhibit substantial particle-induced dynamic load imbalance during the course of the calculation.
- Each of these problems is time dependent, and the particle distributions also evolve in space, energy and direction over many cycles.
- Two calculations were made for each of these problems, with the dynamic load balancing feature either disabled or enabled.
- Parallel calculations were run on the MCR machine, a Linux-cluster parallel computer with 2-way symmetric multiprocessor (SMP) nodes at LLNL.

The Results of Dynamic Load Balancing in MERCURY

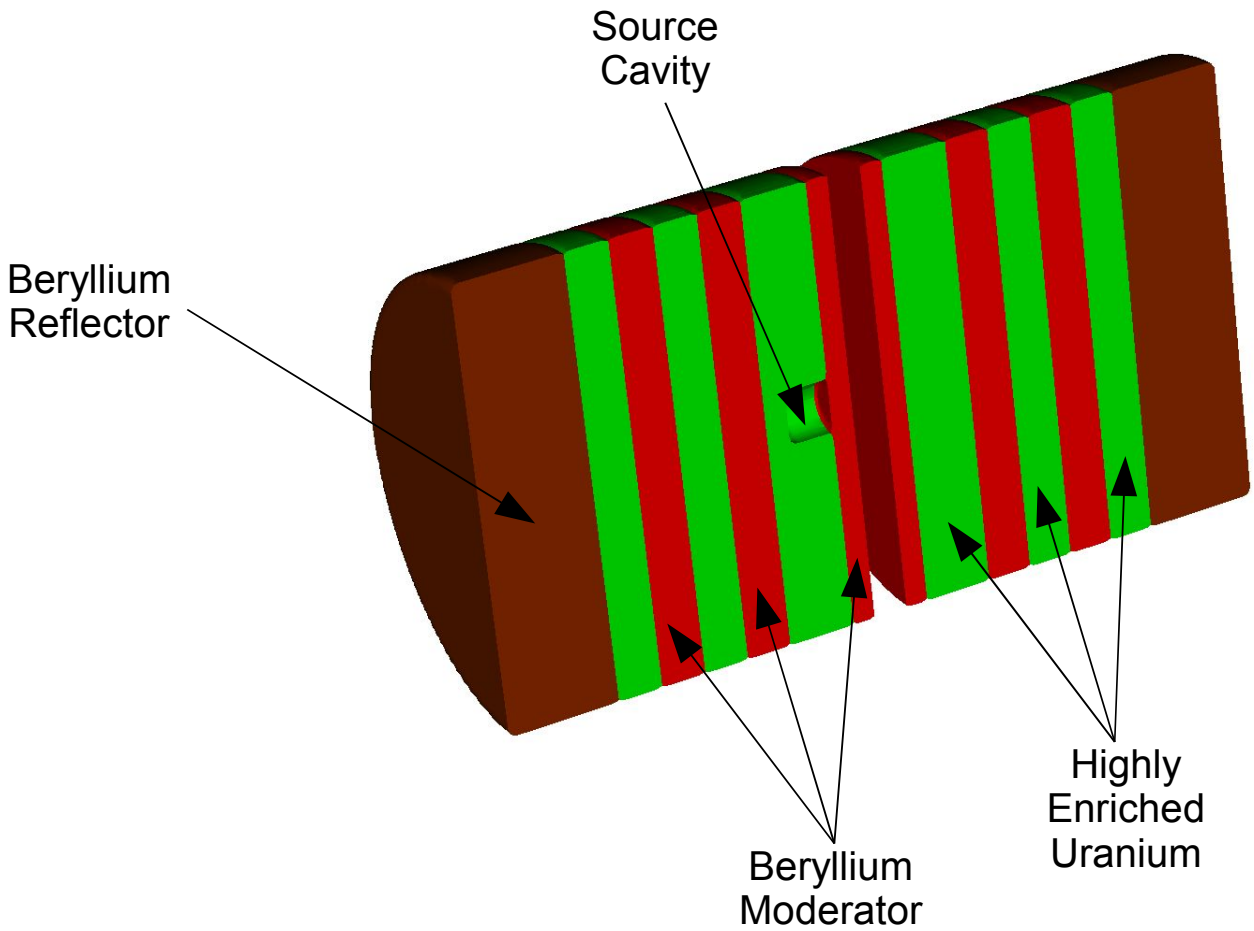
Criticality Test Problem

- The criticality problem is one of the benchmark critical assemblies compiled in the *International Handbook of Evaluated Criticality Safety Benchmark Experiments*.
- The particular critical assembly is a known as HEU-MET-FAST-017: a right-circular cylindrical system comprised of alternating layers of highly-enriched uranium and beryllium, with beryllium end reflectors.
- These calculations were run with $N_p = 2 \times 10^6$ particles, using a "pseudo-dynamic" algorithm that iterates in time to calculate both the k_{eff} and α eigenvalues of the system.
- This problem was run on a 2-D $r - z$ mesh that was spatially decomposed into 14 domains, axially along the axis of rotation. Parallel calculations were run on 28 processors.

The Results of Dynamic Load Balancing in MERCURY

Criticality Test Problem

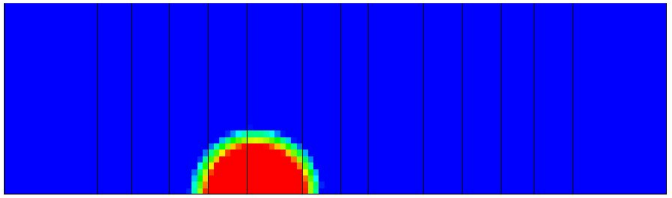
Critical Assembly HEU-MET-FAST-017



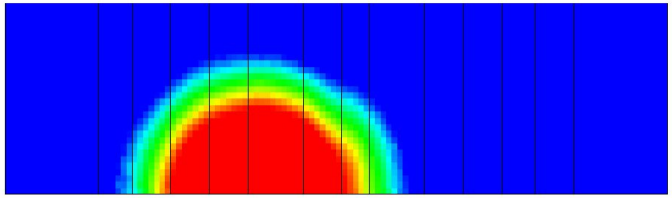
The Results of Dynamic Load Balancing in MERCURY

Criticality Test Problem

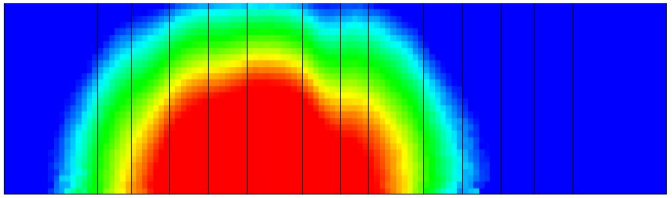
Critical Assembly HEU-MET-FAST-017



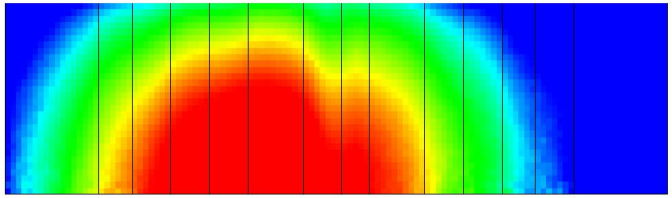
Cycle 1



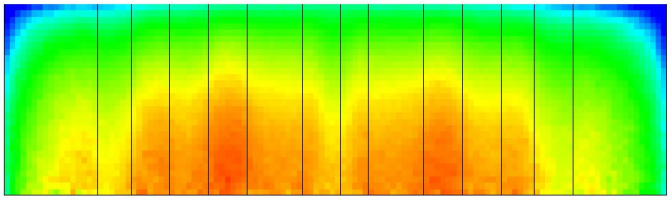
Cycle 2



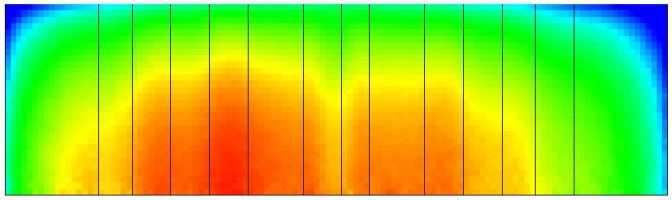
Cycle 3



Cycle 4



Cycle 8



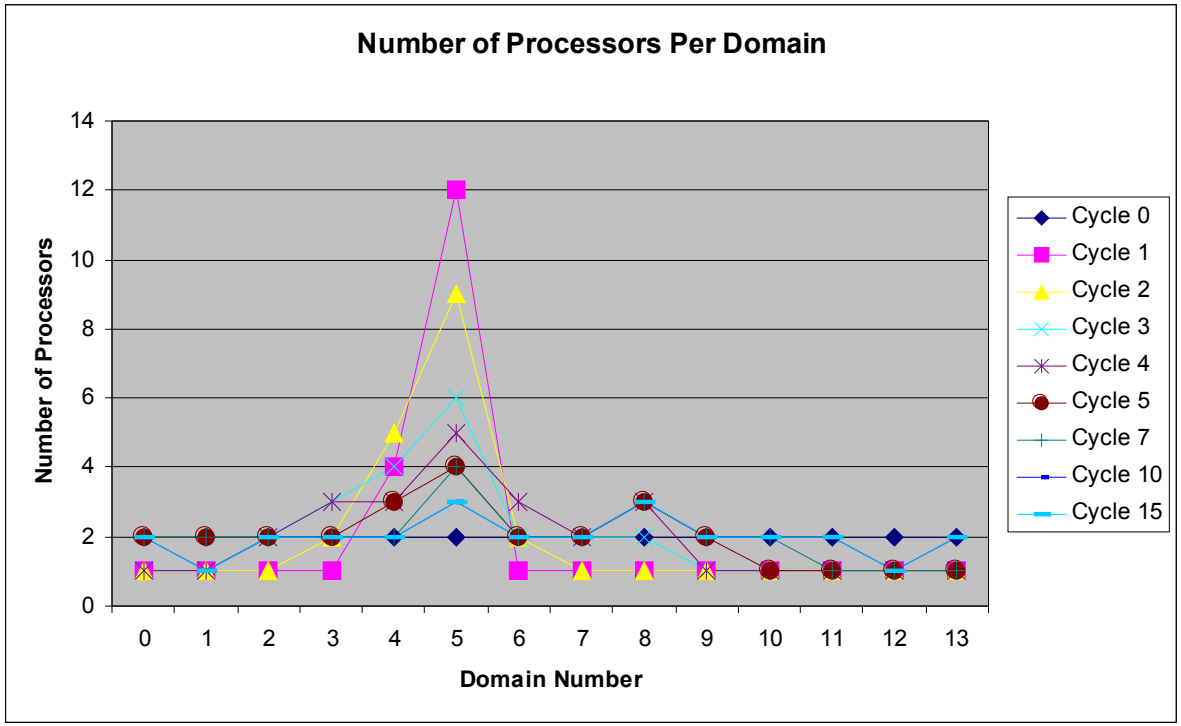
Cycle 15

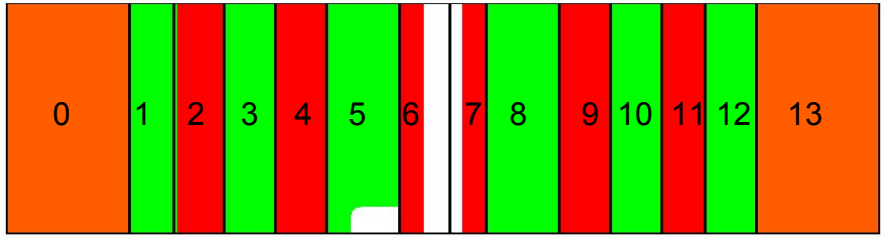
Pseudocolor plots of neutron number density per cell at six cycles during the calculation.
14-way Spatial Parallelism (Black lines represent domain boundaries).

The Results of Dynamic Load Balancing in MERCURY

Criticality Test Problem

Critical Assembly HEU-MET-FAST-017

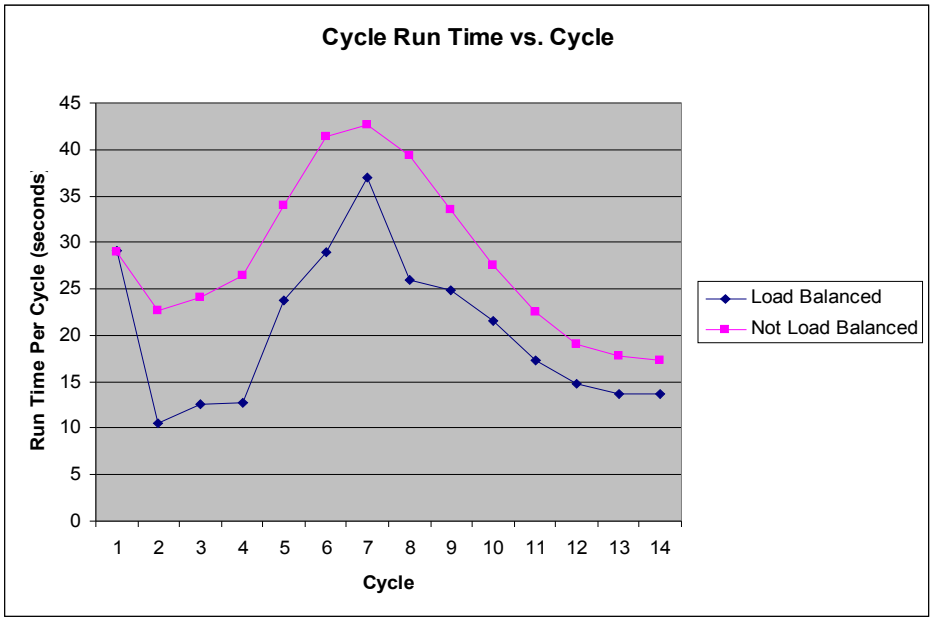


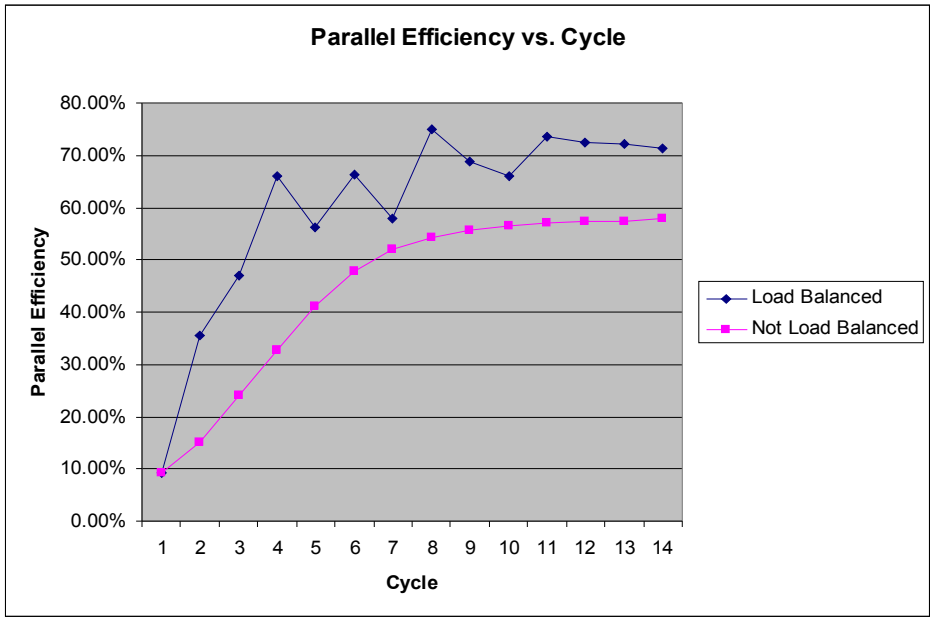


The Results of Dynamic Load Balancing in MERCURY

Criticality Test Problem

Critical Assembly HEU-MET-FAST-017





Critical Assembly Test Problem Cumulative Run Times

Problem Run Time (sec)			
Cycle Range	*Not Load Balanced*	*Load Balanced*	*Speedup*
1 to 4	102.2	65.0	1.57
1 to 14	397.1	286.4	1.39

The Results of Dynamic Load Balancing in MERCURY

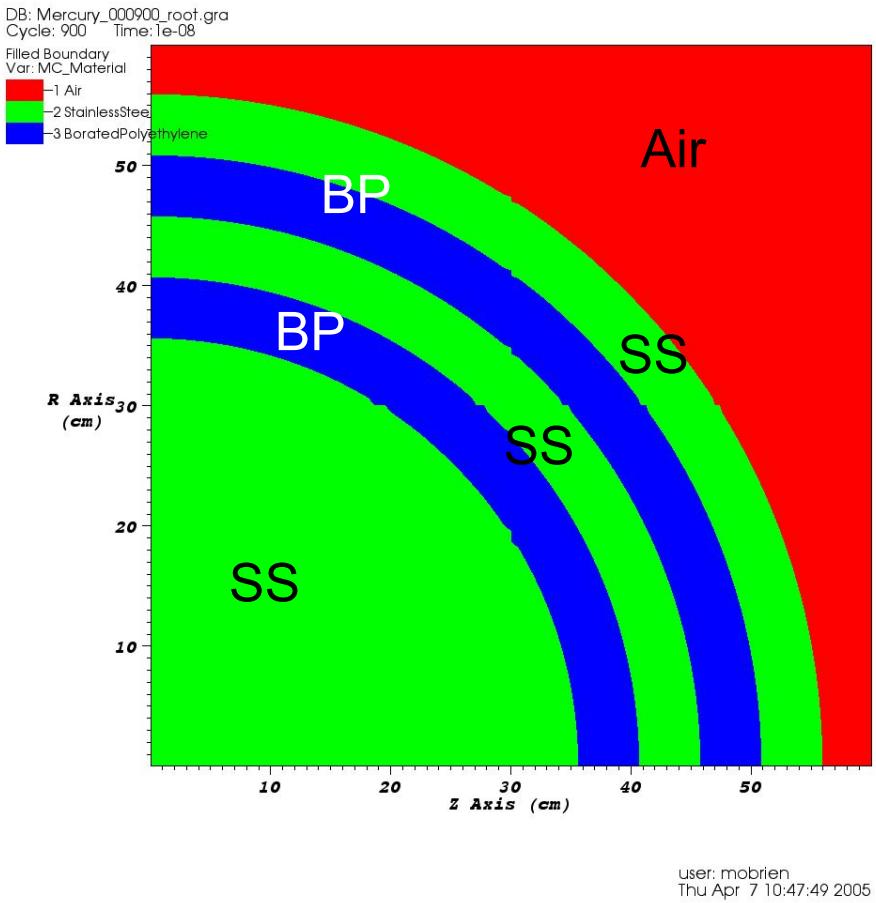
Sourced Test Problem

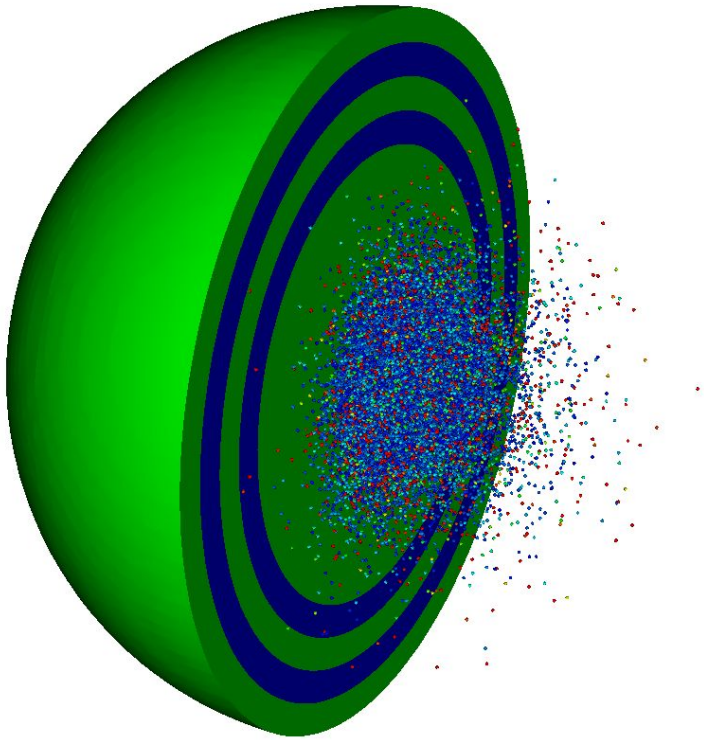
- The time-dependent sourced problem is a spherized version of a candidate neutron shield that would surround a fusion reactor.
- The shield consists of alternating layers of stainless steel and borated polyethylene.
- Monoenergetic $E = 14.1$ MeV particles are sourced into the center of the system from an isotropic point source.
- During each of the the first 200 cycles, $N_p^{'} = 1 \times 10^5$ particles are injected into the system. The source is then shut off, and the particles continue to flow through the shield for the next 800 cycles. The size of the time step is $\Delta t = 1 \times 10^{-8}$ sec.
- This problem was run on a 2-D $r - z$ mesh that was spatially decomposed in 4 domains, 2 domains along each of the axes. Parallel calculations were run on 16 processors.

The Results of Dynamic Load Balancing in MERCURY

Sourced Test Problem

Spherical Neutron Shield

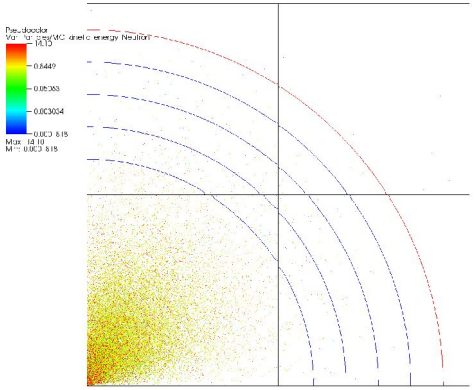




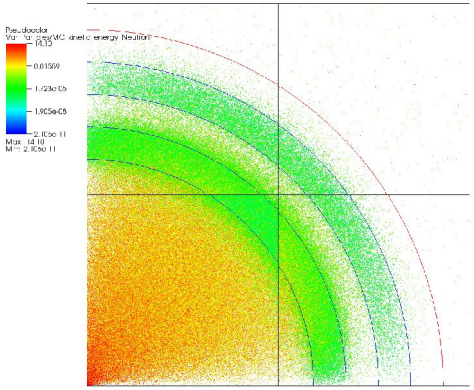
The Results of Dynamic Load Balancing in MERCURY

Sourced Test Problem

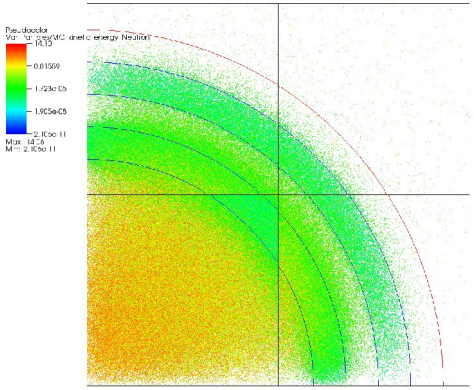
Spherical Neutron Shield



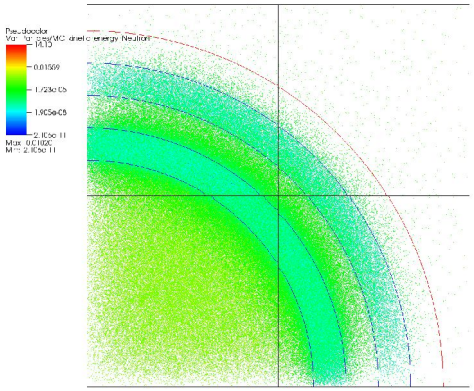
Cycle 2



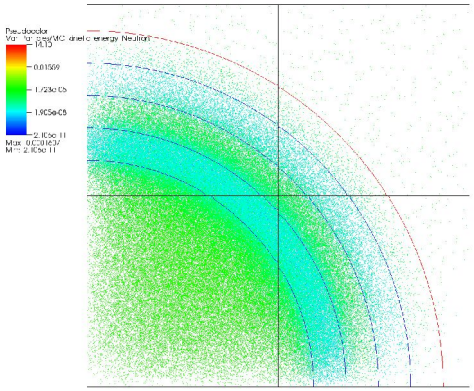
Cycle 101



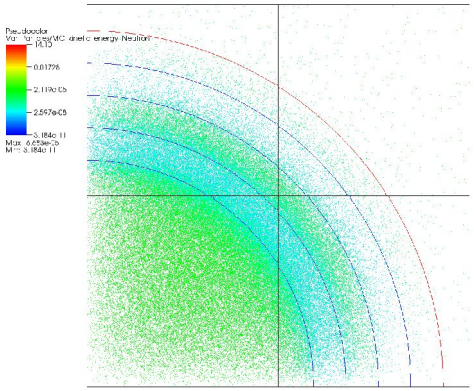
Cycle 201



Cycle 301



Cycle 701



Cycle 1001

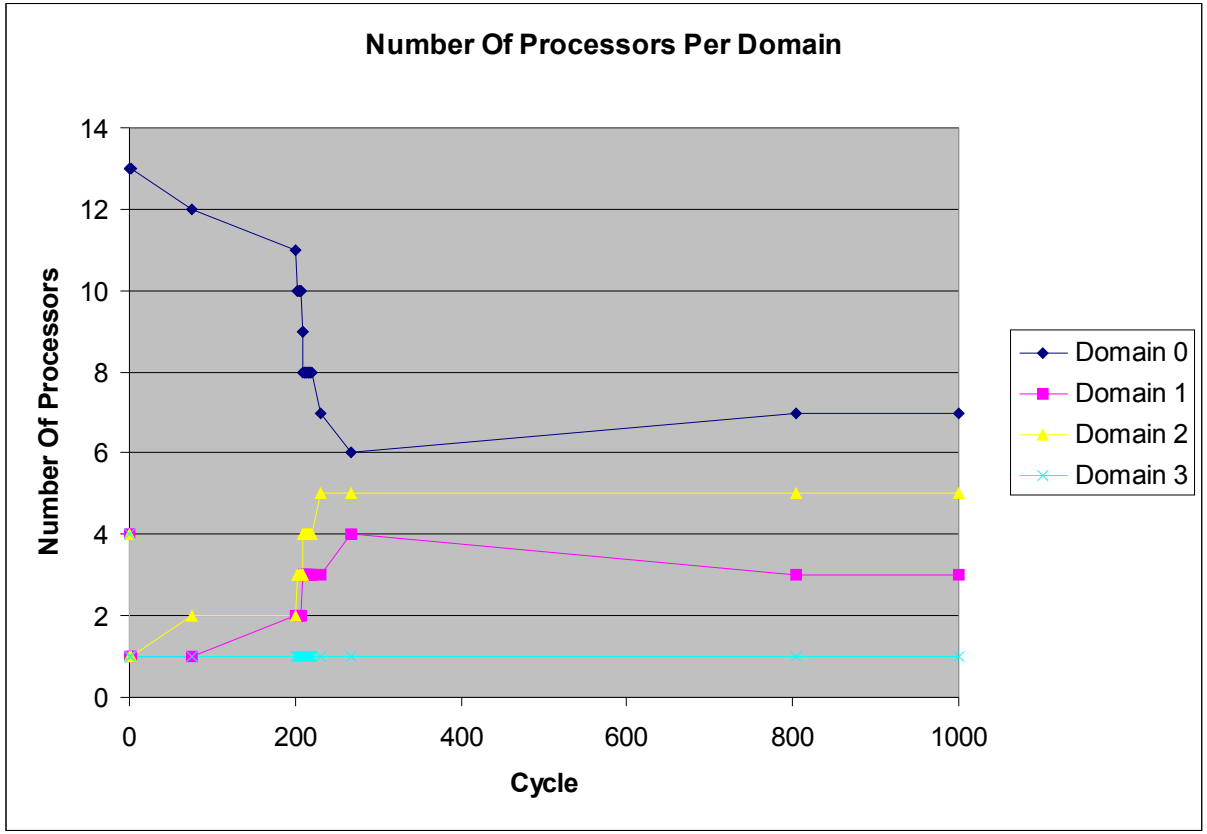
Particle scatter plots of neutron kinetic energy at six cycles during the calculation.

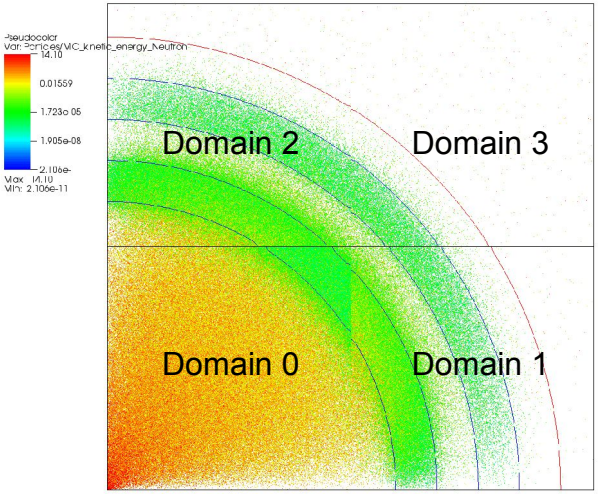
4-way Spatial Parallelism (Black lines represent domain boundaries).

The Results of Dynamic Load Balancing in MERCURY

Sourced Test Problem

Spherical Neutron Shield

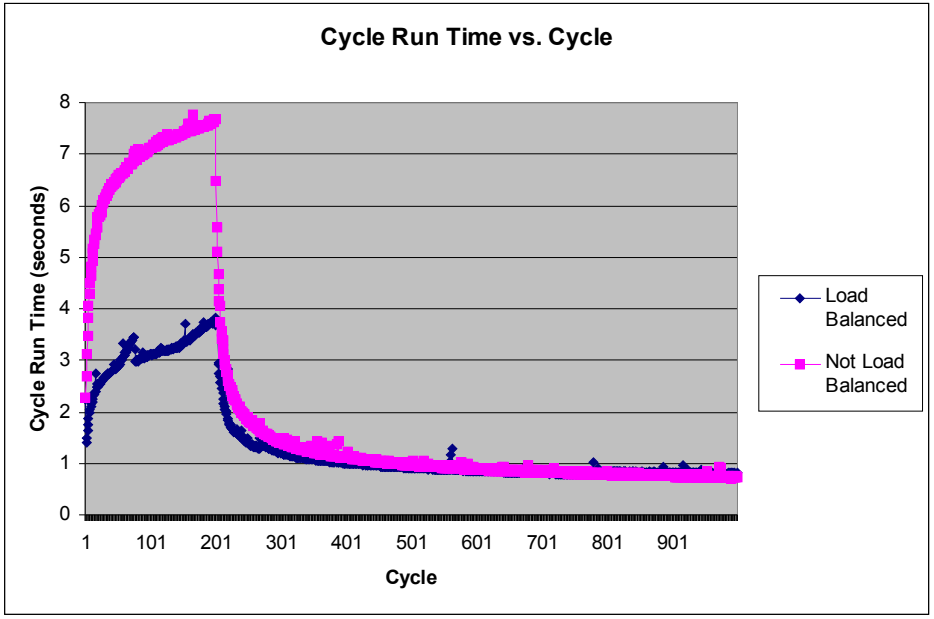


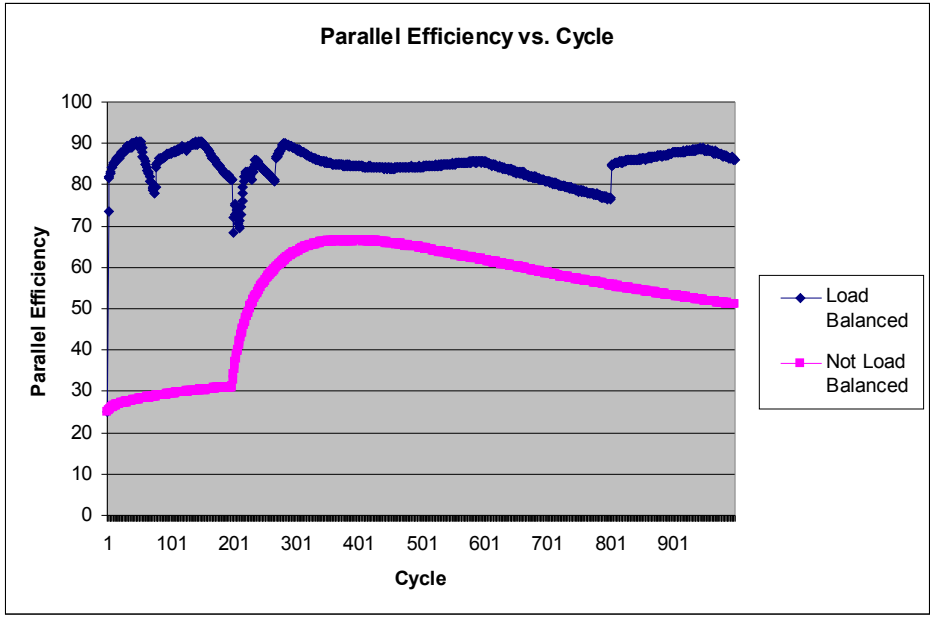


The Results of Dynamic Load Balancing in MERCURY

Sourced Test Problem

Spherical Neutron Shield





Spherical Shield Test Problem Cumulative Run Times

Problem Run Time (sec)			
Cycle Range	*Not Load Balanced*	*Load Balanced*	*Speedup*
1 to 201	1355	615	2.20
1 to 1001	2221	1404	1.58

Summary and Conclusions

- Particle-induced load imbalance in Monte Carlo transport calculations is a natural consequence of a parallelization scheme which employs spatial domain decomposition.
- The synchronous nature of several Monte Carlo algorithms means that the parallel performance of the overall calculation is limited by the performance of the processor with the most loaded domain.
- A load balancing method has been developed and implemented in MERCURY for use in parallel Monte Carlo calculations which employ both domain decomposition (spatial parallelism) *and* domain replication (particle parallelism).
- This method varies the replication level of each domain dynamically in response to the work (number of particle segments) on that domain.
- When the load balancing method has been enabled during parallel calculations of a criticality problem and a sourced problem, the parallel efficiency of MERCURY has been observed to increase by *more than a factor of 2*.

For additional information, please visit our web site:

www.llnl.gov/mercury

Acknowledgments

This work was performed under the auspices of the

U. S. Department of Energy by the University of California

Lawrence Livermore National Laboratory

under Contract W-7405-Eng-48.