## ABSTRACT

HAYS, ROSS DANIEL. Boiling Water Reactor In-Core Fuel Management through Parallel Simulated Annealing in FORMOSA-B. (Under the direction of Paul J. Turinsky.)

A commercial nuclear power plant with a boiling water reactor will utilize between 368 and 800+ individual fuel assemblies to generate steam for 18 to 24 months between refueling outages. The composition and reactivity of each fuel assembly will vary due to variations in initial enrichment, burnable poison loading and irradiation conditions in the core. These variations pose a challenge to the engineers who must design subsequent reloads because only one quarter to one half of the fuel will be replaced at a time. One of the challenges is to determine the optimum layout of the fuel within the core in order to get the highest value from the fuel without violating any safety or operational limits. The `FORMOSA-B` program [23] was developed to automatically find a family of optimum loading patterns by combining a robust, accurate 3-D core simulator with a simulated annealing loading pattern search. Other features have been added to allow the program to rapidly compute shutdown margins [12] and optimize control rod programming through the application of heuristic rules [11]. One drawback of the `FORMOSA-B` program is that long run-times, sometimes exceeding a week, are required to generate and evaluate the large numbers of solutions required by the simulated annealing algorithm. The rising popularity and availability of parallel computing and computational clusters provides a possible solution to the problem of long run-times. To this end, a parallel simulated annealing capability has been developed for the `FORMOSA-B` program.

The parallel simulated annealing driver utilizes standardized Message Passing Interface routines to divide the individual Markov search chains of the simulated annealing algorithm among a large number of processors. By evaluating multiple loading patterns concurrently, run times are significantly reduced. In testing with a 368-assembly BWR/4 model, parallel speedup factors exceeding 32 were observed with 48 processors. Parallel efficiencies are calculated to be in the range of 68% to 95% when correcting for hardware variations and CRP update frequency. Further testing was performed to investigate the effects on the annealing performance of the Control Rod Programming update frequency, Markov chain length versus parallelization width and solution downselect method.

المنارة للاستشارات
www.manaraa.com

Boiling Water Reactor In-Core Fuel Management through Parallel Simulated Annealing in FORMOSA-B

by
Ross Daniel Hays

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fullfillment of the
requirements for the Degree of
Master of Science

Nuclear Engineering

Raleigh, North Carolina

2009

APPROVED BY:

Dr. Edward Davis

Dr. Robert White

Dr. Paul J. Turinsky
Chair of Advisory Committee

المنارة للاستشارات
www.manaraa.com

## DEDICATION

**Hofstadter's Law:** It always takes longer than you expect, even when you take Hofstadter's Law into account.

*-Douglas Hofstadter: Gödel, Escher, Bach: An Eternal Golden Braid, 20th anniversary ed., 1999, p. 152.*

المنارة للاستشارات
www.manaraa.com

# BIOGRAPHY

Ross Hays was born to Carl and Billie Hays in 1980 in Iowa City, Iowa. After graduating from Burlington Community High School in 1999, he enrolled at Northwestern University, earning bachelors degrees in applied mathematics and chemical engineering in 2004. From there he moved to Raleigh, North Carolina to pursue Masters and PhD degrees in nuclear engineering at North Carolina State University under the direction of Professor Paul J. Turinsky.

المنارة للاستشارات
www.manaraa.com

## ACKNOWLEDGMENTS

The long-awaited completion of this project would not have been possible without the assistance of a great many people. First, I would like to thank my advisor, Dr. Paul Turinsky, for his patient guidance and support through several years of setbacks and delays. Additionally I would like to thank Dr. Paul Keller for the guidance on the gory details of the `FORMOSA-B` code, the advice on the proper choice of a text editor and the introduction to flying.

The completion of this project would also not have been possible without the technical assistance of Dr. Eric Sills and Dr. Gary Howell at the NC State High Performance Computing Center.

I would also like to thank my friends and coworkers here in the NE department for their helpful advice, thoughtful commentary and timely distractions. In particular, Drs Matthew Stokely, Matthew Jessee, Loren Roberts, William Wieselquist and Mr. Robert Newnam.

Finally, I wish to thank my family for their constant support and encouragment.

المنارة للاستشارات

www.manaraa.com

# TABLE OF CONTENTS

**LIST OF TABLES** . . . . . vi

**LIST OF FIGURES** . . . . . vii

**1 Introduction** . . . . . 1
- 1.1 Overview of BWR In-Core Fuel Management . . . . . 3
  - 1.1.1 Problem Description . . . . . 3
  - 1.1.2 FORMOSA-B . . . . . 5
  - 1.1.3 Simulated Annealing, A Brief History . . . . . 7
- 1.2 Parallel Computation . . . . . 10
  - 1.2.1 Commercial Cluster Hardware . . . . . 12
  - 1.2.2 Message Passing Interface Standard . . . . . 13
- 1.3 Performance Metrics . . . . . 14
- 1.4 Parallel Optimization Algorithms . . . . . 15
  - 1.4.1 Parallel Simulated Annealing . . . . . 16

**2 MMCPSA** . . . . . 19
- 2.1 Cooling Schedules . . . . . 19
- 2.2 Synchronous SA . . . . . 20
- 2.3 Asynchronous SA . . . . . 25
- 2.4 Interprocess Coordination and Communication . . . . . 26

**3 Numerical Results** . . . . . 28
- 3.1 Testing Environment . . . . . 28
- 3.2 Test Cases . . . . . 29
- 3.3 Performance Metrics . . . . . 30
- 3.4 Benchmarks . . . . . 32
- 3.5 Tuning the Parallel Algorithm . . . . . 34
  - 3.5.1 CRP Update Frequency . . . . . 35
  - 3.5.2 Depth versus Width . . . . . 42
- 3.6 Stochastic versus Deterministic Branching . . . . . 48
- 3.7 Parallel Performance . . . . . 53
  - 3.7.1 Communications Overhead . . . . . 53
  - 3.7.2 Memory Usage . . . . . 53

**4 Conclusions and Recommendations** . . . . . 55
- 4.1 Future Work . . . . . 56

**BIBLIOGRAPHY** . . . . . 57

المنارة للاستشارات
www.manaraa.com

## LIST OF TABLES

Table 3.1 368-Assembly Quarter-Core Loading Pattern ............ 29
Table 3.2 Optimization Test Case Settings ............ 30
Table 3.3 Serial Benchmark Results ............ 32
Table 3.4 Benchmark Comparison Results ............ 34
Table 3.5 CRP Speedup Results ............ 36
Table 3.6 CRP Optimization Results ............ 37
Table 3.7 Depth versus Width Settings ............ 42
Table 3.8 Depth versus Width Speedup Results ............ 43
Table 3.9 Depth versus Width Optimization Results ............ 43
Table 3.10 Binary Branching Settings ............ 48
Table 3.11 Binary Branching Speedup Results ............ 48
Table 3.12 Binary Branching Optimization Results ............ 49

المنارة للاستشارات
www.manaraa.com

## LIST OF FIGURES

Figure 2.1 PSA Flowsheet ... 20

Figure 2.2 Global Update Logic ... 21

Figure 2.3 Synchronous Parallel Simulated Annealing Program Flow ... 23

Figure 2.4 Set Update Flags ... 24

Figure 2.5 Asynchronous Update Request Handling ... 25

Figure 3.1 CRP Markov-Chain Averaged Objective Function ... 38

Figure 3.2 CRP Markov-Chain Averaged MFLPD ... 38

Figure 3.3 CRP Markov-Chain Averaged MAPRAT ... 39

Figure 3.4 CRP Markov-Chain Averaged MFLCPR ... 39

Figure 3.5 CRP Markov-Chain Averaged CSDM ... 40

Figure 3.6 CRP Markov-Chain Averaged Max HX ... 40

Figure 3.7 CRP Markov-Chain Averaged Flow Violation ... 41

Figure 3.8 Depth versus Width Markov Chain Averaged Objective Function ... 44

Figure 3.9 Depth versus Width Markov Chain Averaged MLFPD ... 45

Figure 3.10 Depth versus Width Markov Chain Averaged MAPRAT ... 45

Figure 3.11 Depth versus Width Markov Chain Averaged MFLCPR ... 46

Figure 3.12 Depth versus Width Markov Chain Averaged CSDM ... 46

Figure 3.13 Depth versus Width Markov Chain Averaged Max HX ... 47

Figure 3.14 Depth versus Width Markov Chain Averaged Flow Violation ... 47

Figure 3.15 Binary Branching Markov Chain Averaged Objective Function ... 49

المنارة للاستشارات

www.manaraa.com

Figure 3.16 Binary Branching Markov Chain Averaged MFLPD ........................ 50

Figure 3.17 Binary Branching Markov Chain Averaged MAPRAT ..................... 50

Figure 3.18 Binary Branching Markov Chain Averaged MFLCPR ...................... 51

Figure 3.19 Binary Branching Markov Chain Averaged CSDM ......................... 51

Figure 3.20 Binary Branching Markov Chain Average Max HX ......................... 52

Figure 3.21 Binary Branching Markov Chain Average Flow Violation ................. 52

المنارة للاستشارات

www.manaraa.com

# Chapter 1

# Introduction

The operation of a commercial Light Water Reactor (LWR) power reactor involves the controlled conversion of energy stored in the nucleus of a fissile isotope (usually Uranium-235 or Plutonium-239) into electricity for sale to consumers. This conversion process occurs though a self-propogating chain reaction in which neutrons cause the fissile nucleus to split, emitting further neutrons and energetic fission fragments. The neutrons go on to split other fissile nuclei, releasing further neutrons. Through this process fissile isotopes are consumed and radioactive daughter nuclei are produced. While certain of these daughter isotopes and their decay products are neutronically inert, a number of them are strong neutron absorbers. The gradual accumulation of these so-called poisons in the fuel, along with fissile isotope depletion and mechanical wear, limit the usable lifetime and energy value of the individual fuel assemblies. Once the fuel assembly has reached its useful life span it is removed from the reactor and replaced with a fresh assembly. For these purposes, nuclear fuel management is the process by which engineers specify the loading of fuel and control elements within a reactor to meet safety requirements and economic goals.

In a batch-operated[1], commercial reactor setting this task can be quite complex and is generally separated into several levels. The first level is the Out-of-Core Fuel Management problem, where one estimates the core-average fissile enrichment required to meet the energy demands of the upcoming fuel cycle. This estimate will also specify which of

[1]All power reactors in the United States operate on a batch-basis, requiring partial disassembly of the reactor vessel in order to change out the fuel. This is in contrast to continuously fueled reactors such as the CANDU, developed by AECL of Canada.

www.manaraa.com

the used fuel assemblies are to be carried over to the next cycle and what should be the average enrichment of the new assemblies. Cycle length and power production estimates are based on the operational experiences for a given unit and on grid-wide load forecasts. Because nuclear power plants generally have the lowest marginal operating costs, they are scheduled at the maximum available output with refueling outages occurring at periods of minimum demand (typically the spring or fall, when air-conditioning and heating loads are at their lowest). Given the electrical energy to be produced from a given load of fuel, it is a simple matter to compute the cycle burnup, which is defined as the total thermal output (in gigawatt-days) per ton of initial uranium loading (or heavy metal, if several fissile species are present). Fissile depletion and mechanical wear in a fuel assembly are closely tied to the amount of burnup that the assembly has undergone; thus this is a limiting factor in determining the usable lifetime for each assembly. Once the burnup of the upcoming cycle is known, the existing reactor fuel inventory may be examined to see which assemblies have sufficient margin remaining for one more cycle. The fresh fuel assemblies to be added in the upcoming cycle comprise one region, while those first loaded and retained from the previous cycle make up another. There will be an additional region for each additional reload worth of fuel in the core at any time. A typical large BWR core may have between two and four fuel regions. Each region can be further separated into batches, such that all assemblies in the batch have identical irradiation histories (i.e. cycles irradiated in the reactor) and fissile enrichments. Once the sizes of the new batches of fuel have been determined, the engineer must estimate the required fissile enrichment of each batch such that the core will remain critical until the end of the cycle.

The next level of the fuel-cycle management problem is In-Core Fuel Management. At this level the location for each fuel assembly within the core and the reactivity control scheme are determined in order to minimize costs while satisfying all operational and thermal limitations. As the typical Boiling Water Reactor core contains between 368 and 800 fuel assemblies (up to 1132 in some new designs) and over 200 control blades, the number of unique combinations and permutations can be quite large (on the order of $10^{100}$[4]). Several methods are available to produce optimum solutions to this problem; these are explored further in the next section.

The lowest level of fuel cycle management is the design of the fuel assemblies

المنارة للاستشارات
www.manaraa.com

themselves. This process, typically performed by the fuel manufacturer, aims to maximize the reliability, economy and safety of the fuel. Designers of fuel bundles have a wide range of parameters with which to vary the properties of a given bundle. These range from large changes such as the number and size of the fuel rods, to smaller changes, such as the design of a spacer grid, or the variation of fissile enrichment and burnable poison loading within an assembly or rod. Each batch in a reload core will have the same bundle design.

## 1.1 Overview of BWR In-Core Fuel Management

### 1.1.1 Problem Description

With the new batch of fuel having been specified by the Out-of-Core design process, it falls to the In-Core fuel management process to determine the optimum arrangement of the individual fuel assemblies within the reactor. The specific arrangement of the assemblies with respect to each other is termed the Loading Pattern (LP). The new loading pattern is chosen to address several concerns, such as the minimization of neutron leakage (for economy) and the maximization of thermal margins (for safety). As in many situations, these turn out to be competing goals. For example, to minimize neutron leakage, it is desirable to depress neutron flux around the periphery of the reactor by loading the least reactive assemblies in the outermost positions. This causes the radial flux shape to be sharply peaked around the middle, leading to high power peaking factors. This cuts into thermal margins, increasing the chances of fatigue related fuel failure and limiting the maximum power-level of the core. Conversely, to flatten the radial power shape (and thereby maximize thermal margins) it is desirable to put the least reactive fuel in the center of the core while putting the more reactive fresh fuel towards the periphery. This increases the neutron leakage, thus requiring higher fissile loadings, increasing fuel costs.

A further safety consideration for the BWR loading pattern is the requirement that the core remain subcritical[2] at cold shutdown conditions (68° F, xenon-free) with the highest reactivity worth control rod completely removed from the reactor. This ensures that should any one control rod fail to insert for any reason, the operators would still be able to bring the plant down to a safe condition. As it is not known beforehand which single

[2] By a margin of at least 0.38% $\Delta k/k$ as per [24, 25]

www.manaraa.com

control blade will have the highest worth at shutdown conditions, one must individually evaluate each of the control elements to determine which one presents the limiting case.

Another consideration unique to BWRs is the sequencing of the control rods. Unlike a Pressurized Water Reactor (PWR), a BWR cannot use soluble boron in the coolant to provide shim reactivity control during operation. However, the BWR can vary the coolant flow rate through the core by upwards of 15 to 20 percent utilizing variable-speed jet pumps in the reactor vessel downcomer. This changes coolant flow rate, which alters the fraction of steam (void) within the core. Because the core is undermoderated by design, a loss of coolant density causes a decrease in reactivity, decreasing the power level and causing a partial reduction in void fraction. The overall effect is that by increasing the coolant recirculation rate, one can achieve a corresponding increase in core power without moving any control blades. Alternatively, flow control can be used to keep the reactor operating at a fixed power level by offsetting the reactivity drop associated with fuel burnup.

Unfortunately, flow control cannot provide sufficient negative reactivity to allow the reactor to operate with all control blades fully withdrawn during the cycle. The BWR must instead rely on a changing combination of control blades and recirculation flow control throughout the cycle; this combination of control blade insertions and recirculation flow is known as the Control Rod Program (CRP).

There are two main tasks that a Control Rod Program must accomplish. First, it must keep the power profile somewhat flat, both axially and radially, so that thermal limitations are satisfied at full power. Second, the control rod program must minimize the negative effects of burnup shadowing. Burnup shadowing occurs when fuel assemblies adjacent to a control blade operate at lower power than surrounding assemblies. Thus they accrue less burnup and retain more of their original reactivity. When the control rod is withdrawn the highly reactive fuel causes the power density in that location to rise sharply. This rapid change in power density can lead to Pellet Clad Interaction (PCI) failure, whereby the thermal expansion of the fuel pellet causes a rupture of the cladding material.

Two heuristic strategies have evolved for selecting a control rod program to minimize burnup shadowing effects. The first strategy is the Control Cell Core (CCC) strategy, whereby a subset of the control rods (typically one in four) and the four adjacent fuel assemblies per control rod are designated as Control Cells. As the cycle progresses, the

المنارة للاستشارات
www.manaraa.com

control rods are axially positioned within each control cell in such a manner as to maximize the operating thermal margins while holding the core critical. The loading pattern is then constrained such that high-reactivity fuel assemblies cannot be placed face-adjacent within a control cell, and fresh fuel cannot be placed in a control cell at all. Furthermore, fuel is not allowed to stay in a control-cell for more than one cycle. This minimizes burnup shadowing and peaking effects when the control rod movement occurs. The downside of the control cell core is that the fuel placement constraints limit the number of positions available for fresh fuel. This restriction is a limiting factor for reactors running with extended 24-month cycles or going through power uprates, as they require more fresh fuel than can be accommodated using a CCC approach. For these cores a conventional core (COC) strategy is employed.

In a COC control rod program, a subset of all control rods are grouped into four banks, labeled `A-1` ,`A-2` ,`B-1` and `B-2`. At any given timestep in the CRP, only one bank will have deeply inserted control rods. Another bank will be partially inserted for axial power shaping purposes and to achieve core criticality. At regular intervals (approximately five times per annual cycle[28]), the deep rod insertions are shifted from one bank to the next. The rod swap intervals are chosen such that minimal burnup shadowing accrues between each swap, thus minimizing the stress to the fuel cladding. However, even with frequent exchanges it is necessary to perform these rod swaps at a reduced power level in order to ensure cladding integrity.

One final consideration in developing a control rod program is the effect of partially inserted control blades on the axial power shape within the core. In current BWR designs all control blades enter through the bottom of the reactor. Thus a partially inserted control blade will shift the axial flux peak towards the top of the core whereas a fully inserted blade will have very little effect on the axial power shape. For this reason a combination of deep and shallow rod insertions are generally preferred over intermediate insertions when feasible.

### 1.1.2 FORMOSA-B

`FORMOSA-B`, like its predecessor `FORMOSA-P` (for PWRs), was developed at the Electric Power Research Center of North Carolina State University as a computational tool for engineers to aid in solving the In-Core fuel management problem[16, 22, 23, 12, 13]. It uti-

المنارة للاستشارات
www.manaraa.com

lizes an adaptive Optimization by Simulated Annealing (OSA, described below) algorithm to find optimal loading patterns for fresh and reload fuel based on one of several objectives and an array of constraints. Possible objective functions include the maximization of end-of-cycle (EOC) reactivity (or equivalently, the minimization of EOC coolant flow), the maximization of the critical power ratio (a thermal limit relating to cladding failure), minimization of peak linear power density, the maximization of the region-averaged fuel discharge burnup and the minimization of reload fuel cost. Additionally, `FORMOSA-B` has the capability to determine a control rod program by the application of heuristic rules for both conventional and control-cell cores[11].

A number of limits and constraints can be imposed on the optimization search as well, such as: 1) Maximum discharge burnup in a specified batch, region or fuel assembly, 2) Constraints on fuel shuffling symmetries, such as quarter core symmetry or fixed fuel locations, 3) Limits on thermal margins, 4) Constraints on allowable coolant flow rates, 5) Maximum and minimum allowed hot-excess reactivity and 6) Minimum cold shutdown margin. While some of these limits also appear as possible optimization objectives, one never activates a given constraint if it is the current objective, as that would effectively double-count that parameter. The constraints can be categorized as either soft or hard constraints, with the hard constraints being further subdivided into physical constraints and true-false constraints. Physical constraints are those that are satisfied automatically by the requirements of symmetry and logical consistency. For example, a single fuel assembly may only be placed in each location for a given cycle, or fuel loading pattern may be constrained to have certain symmetries. The satisfaction of these constraints is automatic and does not require the evaluation of core neutronics or thermal-hydraulic properties. This is not the case for true-false constraints, where one does not know *a priori* whether a given pattern will be satisfactory. (Thermal margins, for example, would fall into this category.) By contrast, a soft constraint does not result in the automatic rejection of a candidate solution. Instead, the degree by which the calculated value exceeds the constraint is multiplied by an adaptively computed penalty factor and is then added to the overall 'penalized' objective function for the solution. The result is that solutions exhibiting violations of the soft constraints may be accepted, but with a probability that reduces in proportion to the violation severity. This important feature enhances the search algorithm's ability to traverse through infeasible

المنارة للاستشارات
www.manaraa.com

areas of the configuration space and escape from local minima that may be present.

As will be discussed later, the use of the OSA optimization algorithm requires the repetitive evaluation of a large number of possible configurations. Each pattern evaluation requires solving the full 3-D coupled neutronics and fluid equations at each timestep. For this reason `FORMOSA-B` utilizes a fast core simulator based on the Nodal Expansion Method[22] to quickly yet accurately model the key core properties related to the varied safety, economic and operational limits imposed on the design. Even with a fast core simulator the vast number of simulations required by the stochastic search algorithm severely limits the practical usability of the program on a single workstation. Shortening these long computational turnaround times is the primary motivation behind the parallelization efforts presented here. Before delving into the details of an optimization by parallel simulated annealing methodology, it is prudent to examine both the theoretical underpinnings of the simulated annealing algorithm and the available parallel computation resources.

### 1.1.3 Simulated Annealing, A Brief History

The simulated annealing algorithm was originally developed in the 1950's by Metropolis et al [21, 30] as a fast method for calculating equations-of-state for various materials and conditions. It utilizes a modified Monte Carlo integration over configuration space to solve statistical mechanics problems that are not soluble analytically. This algorithm is ideally suited to optimization problems with large combinatorial search spaces and no readily available differential information. In many cases, the bulk of the computational burden lies in the evaluation of the optimization objective function, and not in the generation of new, perturbed configurations.

In the metallurgical process of annealing, a sample is heated to the point where the atoms in the lattice have sufficient thermal energy to overcome local potential barriers and shift positions. Each atomic shift alters the amount of potential energy stored in the lattice; the individual atomic displacements are stochastic in nature and may increase or decrease the amount of energy stored in the lattice structure. The overall trend is such that the lattice preferentially moves towards a low energy configuration. The sample is then slowly cooled; this causes the individual displacements to occur less frequently and reduces the probability of any transition occurring that would increase the potential energy

المنارة للاستشارات
www.manaraa.com

of the lattice. The rate at which the sample is cooled plays a large role in determining its final configuration. The more slowly it is cooled, the more likely it is that it will reach a minimum energy state (e.g. perfect crystals or large grain sizes). If it is cooled too quickly (i.e. *quenched*), then the lattice configuration will be fixed in a higher energy state. To summarize, the process of annealing alters the atomic configuration of a sample in a way that tends to minimize the amount of potential energy stored within its structure.

In simulated annealing, one seeks to find the minimum value of a given *Objective Function* by manipulating the configuration of the input variables of said function. Computationally the simulated annealing algorithm proceeds through the repeated evaluation of a series of perturbed input variable configurations. At each iteration, the set of input variables is randomly changed in some small manner from a reference configuration. If this new configuration results in a lower objective function value than the previous configuration, it is adopted as the reference solution, and the process is repeated. If, however, the new configuration results in a higher objective function value than the reference case, it will be accepted with a probability given by the *Metropolis Criterion* [30]:

$$p = e^{-\Delta E/T} \tag{1.1}$$

where $\Delta E$ is the change in objective function from the reference case and $T$ is the *Simulated Annealing Temperature* (described below). Given a sufficient number of repetitions, the distribution of the objective function values will approach an equilibrium Boltzmann distribution [30]:

$$P(E) = \frac{1}{Z(T)} e^{-E/k_B T} \tag{1.2}$$

where $T$ is the *temperature* of the system, $E$ is the objective function value (which corresponds to energy in the statistical mechanics analogy), $k_B$ is the Boltzmann constant (which relates energy to temperature), and

$$Z(T) = \sum_{i \in S} e^{-E_i/k_B T} \tag{1.3}$$

is the partition function (where $S$ is the so-called configuration space, spanning all possible solutions) . By examining equation 1.2, it can be seen that as $T$ approaches zero at equilibrium, the probability that the configuration is in any state but the minimum energy state vanishes. Thus, by slowly lowering the temperature parameter in the algorithm, we

للاستشارات المنارة
www.manaraa.com

are guaranteed of asymptotic convergence to the global minimum solution. The choice of this so-called *cooling schedule* is a key factor in determining the performance of the SA algorithm.

As with physical annealing, the proper selection of the cooling rate will largely determine the quality of the final outcome. Unfortunately, as can be expected of any stochastic algorithm, convergence to a globally minimum solution can only be rigorously proven in the limit of an infinite number of simulations. The Freidlin-Wentzell Framework[2] predicts that for a certain set of general conditions

$$P(y_n \notin E_0) \leq \left(\frac{K}{n}\right)^{B/A} \tag{1.4}$$

where $P(y_n \notin E_0)$ is the probability that the solution $y_n$ is not in the set of global minima $E_0$; $K$, $B$, and $A$ are positive constants relating to the so-called *energy landscape* of the set of all solutions and the integer $n \geq 1$ is the total number of *cooling steps*, or sets of samples at constant simulated annealing temperature. Note that for this analysis the number of samples for each cooling step $k$ goes as $L_k = c_1 k^{c_2} exp(c_3 k)$ and the SA temperature goes as $T_k = c_4/k$ for given constants $c_1$,$c_2$,$c_3$,$c_4$.

As the simulated annealing temperature changes, so too does the equilibrium distribution of objective function values; however, upon each discrete change in the temperature, a certain number of histories must be sampled before this new distribution is reached. If the temperature is reduced too rapidly, the likely result is *quenching*, whereby the solution becomes trapped in a local minimum. One approach to maintaining a quasi-equilibrium distribution is to follow an *annealing curve*, which relates the average of the objective function to the temperature. This method is used by Huang et al [9] to generate the relation

$$\frac{\mathrm{d}\langle E\rangle}{\mathrm{d}(\ln T)} = \frac{\sigma^2}{T} \tag{1.5}$$

where $\langle E\rangle$ and $\sigma^2$ are the mean and variance of the objective function, respectively. Huang maintains a quasi-equilibrium state by selecting a new temperature such that the expected objective-function average at the new temperature level is less than one standard deviation from the current average value. Thus

$$T_{k+1} = T_k e^{-\lambda T/\sigma} \tag{1.6}$$

المنارة للاستشارات
www.manaraa.com

where $\lambda$ has a recommended value of 0.7, and $T_{k+1}$ should be taken as the greater of the above calculated value or $T_k/2$. This temperature decrement scheme is the default method used by the `FORMOSA-B` code. Huang et al further accelerate the annealing process by monitoring the number of accepted solutions whose objective functions fall within a tolerance, $\delta < \sigma$, of the mean value, the so-called *within count*. If the *within count* exceeds a predetermined fraction of the Markov chain limit, then the system is considered to have reached equilibrium, and temperature is decremented.

Because the simulated annealing algorithm involves a repeating series of trials, it is easy to imagine several different ways to parallelize the algorithm for increased performance. Such parallelization schemes necessarily increase the complexity of both the optimization algorithm and the theoretical analysis thereof. Before examining such issues it is prudent to look into the capabilities, limitations and variations of the modern parallel computing environment.

## 1.2 Parallel Computation

Compared to modern PCs, the first computers seem to be little more than gigantic, power-hungry calculators. However, at the time of their development, they represented a great leap forward over the previous generation of computational power (namely, rooms full of people with adding machines). The size and expense of these early machines initially limited their use to governmental applications such as code-breaking, nuclear weapons calculations and the generation of ballistics tables[29, 26]. The invention of the transistor and the subsequent development of the microchip led not only to vastly greater computational speed and power, but also to greatly reduced size and cost. What used to fill a room and cost hundreds of thousands of dollars, now fits in the pocket, and costs only a few dollars.

Performance improvements have been achieved through a variety of means. First and foremost, advances in materials and fabrication techniques have allowed the speed and number of transistors in a single chip to increase at a nearly exponential pace (Moore's Law[29]). Furthermore, innovations in chip architecture, including instruction-set microcode, multi-level caching, and pipelining have allowed such advances as out-of-order execution, speculative execution, and the execution of multiple instructions per clock cycle.

المنارة للاستشارات
www.manaraa.com

As the price of computer hardware dropped, it became feasible to build computers with multiple processing units. This facilitated a further expansion in calculational power than would be available through chip fabrication advances alone. The ability of a program to effectively run on multiple processors hinges on several key features of both the hardware and the software.

**Parallel Software**

At the software level, the program algorithm must be divisible into distinct, independent tasks that may be executed separately from each other with limited communication. Different algorithms are parallelizable in different ways and to different extents. For some, such as parametric studies and sensitivity analyses, parallelism is obvious and easily achieved; these methods are said to be *embarrassingly parallel*. Certain classes of embarrassingly parallel algorithms may be further broken down into a series of serial computational tasks that may be executed completely independently. Parallel programming is likely to consist largely of automation scripts that simplify the task of managing the large number of input and output files generated. In these algorithms, inter-process communication overhead is generally quite minimal compared to the computational effort required for each process. Others algorithms, such as branching searches or matrix inversion, are capable of parallelization, but require careful analysis of data-flows and dependencies. Data and procedural dependencies will tend to limit the scalability and parallel efficiency (described below) that may be realized. The final class of algorithms are the so-called *serial algorithms*, in which the data and procedure dependencies render them completely unparallelizable. If higher performance is required, it is often necessary to move to an alternative, parallelizable algorithm. The optimization algorithm used for this investigation is very nearly embarrassingly parallel, as will be described below, and is derived from a serial algorithm.

**Parallel Hardware**

A defining characteristic of a parallel computer is the ability to share data between individual processing units. At the hardware level, the computer must have a means of communicating data and instructions between processing units and a means of coordinating said data and scheduling the various operations to be performed. One way such

المنارة للاستشارات
www.manaraa.com

processors can be classified is according to the number of data and instruction streams they are able to process concurrently; the so-called Flynn Taxonomy [29]. At the lowest level are Single-Instruction Single Data (SISD) machines that operate on a single set of data with a single set of instructions. This type encompasses the majority of computers in common use (although multi-core and multi-processor computers have become much more prevalent in recent years.) The next level is the Single-Instruction, Multiple-Data (SIMD) machine. These are commonly known as Vector processors, and are designed to perform identical operations simultaneously on multiple data streams. Although some early super-computers, such as the Cray X-MP utilized vector processors, they have been supplanted by more powerful and versatile Multi-Instruction, Multiple-Data (MIMD) computers. No multiple-instruction, single-data machines have been marketed commercially.

MIMD machines can be further subdivided according to whether the individual processing elements share a single memory, or each have their own address space. The former are known, appropriately enough, as Shared-Memory computers; these range in size and power from simple, multi-core desktop workstations to powerful, high-availability main-frame computers (such as the IBM Z-series [29]). Distributed-Memory machines, however, are generally comprised of groups or clusters of individual computers acting together through a network connection. The individual nodes in a distributed-memory machine may themselves be shared-memory machines or further networks of (real or virtual) distributed-memory machines. Owing to their availability and relative ease-of-use, the distributed memory cluster was the tool of choice for the work to be presented here. Therefore a closer look at their operational features and limitations is warranted.

### 1.2.1 Commercial Cluster Hardware

Traditionally, entry into the world of high-performance computing was greatly restricted by the extremely high cost of the machines and the specialized staff and infrastructure required to support them. This all changed with the advent of the computational cluster. The cluster combines low-cost, widely available computing hardware with specialized management software that allows all nodes to share in the execution of a single program. The number and type of computers to be combined into a cluster is typically quite flexible, and depends primarily upon the intercompatibility of the management and

www.manaraa.com

computation software and the networking hardware. One form-factor that has become quite popular for cluster-computing is that of the compact, rack-mountable unit, as typified by the IBM Blade-Center. In this system, peripheral components such as the power supply, external network connections and interface ports are moved from the computer, known as a 'blade', to a common support chassis. The chassis itself requires seven units of space in a standard 19" rack, and is able to hold up to fourteen blades. The scalability of these designs and their use of widely available commodity components greatly reduce the ownership costs over shared-memory machines of similar performance. The availability of clusters in a wide variety of platforms and configurations is simultaneously an advantage and a disadvantage. It can lend a great deal of power and flexibility, but it can also cause a veritable nightmare of compatibility and interoperability issues. The issues are largely handled through the use of high-level standard interface standards protocols, such as the OpenMP and MPI programming libraries and the Ethernet communication protocols.

### 1.2.2 Message Passing Interface Standard

Computational clusters and supercomputers can be very expensive, but the highly specialized software that runs on them can often be just as, if not more costly to develop. With the rapid pace of hardware development and the resultant short computer lifetime, it was necessary to develop a consistent, high-level communications interface for application developers. One such standard that is of particular import to the work presented here is the Message Passing Interface Standard (MPI) version 1. An implementation of this standard, known as MPICH, and distributed by the Intel Corporation with their Intel Fortran Compiler[10] was utilized in the development of this program.

The MPI standard provides for interface routines for a variety of programming languages, including Fortran, C and C++. These routines provide a simple subroutine-style interface (in Fortran) with which the programmer can send messages between sets of processes. The primary communications routines can be subdivided according to whether they involve individuals or groups of processes (the so-called collective communication methods) and according to whether the subroutine waits for the communication to complete before returning control to the calling program (blocking vs. non-blocking communication). Messages cannot be unilaterally passed from one process to another; for each call to a message

المنارة للاستشارات
www.manaraa.com

sending routine, there must be a matching call to a message receiving routine on the target process. Furthermore, the receiving process must know ahead of time the exact size and type of the message it will be receiving. The first of these issues is efficiently handled through non-blocking communications, while the second can be dealt with either through the use of a predefined message structure or through a multistep communication routine whereby the sending process first communicates the size and type of the message to the receiver then sends the message itself. In addition to identifying the source and destination of each message at both the sending and receiving end, each message is given an integer 'tag' value by the programmer. The tag values must be the same at the sending and receiving process in order for the message to pass through.

Non-blocking receives are of great utility because they can be 'posted' at any time before the message is to be sent, and then checked periodically to see if the message has been received. Thus, rather than having a process wait idly for a message, it is able to continue with other computations until the message is received. Furthermore, this is accomplished without introducing any of the coordination issues typically found in multi-threaded applications development. The following communications coordination pattern is used in this project: All communications messages originate or terminate with the root process[3]. All non-root processes post a single non-blocking receive for control communications, while the root process posts a separate receive for each other process. When it reaches the top of the processing loop, each process checks to see if any control messages are present, and if so, it moves immediately to act on them. It then reposts the receive, and continues with the program.

## 1.3 Performance Metrics

The ultimate measure of performance for any computer program is its ability to increase user's productivity. As there is no clear metric for productivity in the given situation, parallel speedup and efficiency will instead be examined. Parallel speedup is given by the ratio of the wall-clock time of the serial program to the parallel version. Thus a program that runs in 20 seconds in parallel compared to 100 seconds in serial would have

[3] The root process is that process which has been assigned a rank of zero.

للاستشارات المنارة
www.manaraa.com

a parallel speedup of 5. Parallel efficiency is calculated by normalizing the parallel speedup by the number of processors. Thus, if the above example required 10 processors to reach the given speedup, then it would have a parallel efficiency of only 50%. It is often the case that parallel algorithms will exhibit different efficiencies for different numbers of parallel processes involved. This is largely due to the increased fraction of computational effort spent on overhead, such as interprocess coordination and communication.

The calculation of parallel efficiency is complicated somewhat by the nature of the stochastic search algorithm employed. Namely, the stochastic search algorithm does not have a clear-cut endpoint. As the search progresses and the SA Temperature drops, it becomes less and less likely that a less favorable transition will be accepted. Therefore, the efficiency of accepting new solutions (e.g. LPs) drops as the optimization proceeds. To resolve this issue, `FORMOSA-B` has a convergence criteria whereby the search is considered to be converged if one of three criteria are met:

- No transitions are accepted during the latest Markov chain.
- Fewer than 5% of proposed transitions have been accepted during the latest Markov chain AND more transitions than the number of available single-change cases have been attempted since the last reduction in objective function value.
- More than a proscribed number of histories have been evaluated (to limit run-time.)

The stochastic nature of the optimization algorithm further complicates the generation and comparison of performance metrics because the results may vary from one run to the next. To ensure that performance measurements are truly representative of the algorithm, it will be necessary to run the code several times over, each time seeding the random number generator with a different starting value.

## 1.4 Parallel Optimization Algorithms

There exists a vast array of applications in science and industry for optimization methods, so it is no surprise that there also exists a correspondingly wide variety of algorithms to meet the particular challenges of each. Of the different types of algorithms

www.manaraa.com

available, stochastic methods are particularly well suited to the discontinuous, combinatorial nature of the in-core fuel management problem. Two popular stochastic algorithms are Simulated Annealing and Genetic Algorithms[14]. While simulated annealing, as described above, is a computational analogue to the metallurgical annealing process, a genetic algorithm (GA) is a computational analogue to natural selection. In GA, the problem inputs are discretized into individual units, which are then perturbed by a mutation operator to create an initial population. This population is then evaluated based upon a numerical 'fitness' score, and a certain fraction of the population are selected to survive. The population is then regenerated by mutation of the existing individuals and by combining traits of different individuals with a so-called cross-over operator. This process repeats until a termination criteria is met. Because it involves repeatedly calculating the fitness of a population of individuals, genetic algorithms are ideal candidates for parallelization. Simulated Annealing, as originally implemented, involves an ever evolving Markov chain of individual states. Although this limitation to a single search chain makes parallelization more difficult, a number of possible methods have been put forward.

### 1.4.1 Parallel Simulated Annealing

Lee and Lee[19] subdivide the world of parallel simulated annealing into methods involving a single Markov chain, and methods following multiple Markov chains (SMC PSA and MMC PSA, respectively). In the SMC PSA schemes, either the evaluation of a single perturbed pattern is performed in parallel, or several perturbations are analyzed concurrently, with only a single perturbation being carried forward to the next iteration. The interprocess communication required at each step greatly limits the ability of these SMC PSA methods to scale efficiently to large numbers of processes. In order to cut down on communication overhead and allow greater scalability the individual processes are allowed to follow independent search paths. Lee and Lee describe three different MMC PSA schemes, each with varying levels of interprocess communication. The first is the so-called Non-interacting MMC scheme, whereby each process performs a complete SA cooling cycle. Once all processes have finished, the best solution is chosen from amongst the processes. This is functionally no different than performing multiple serial optimizations consecutively with a different search path being taken each time. The next step up in

المنارة للاستشارات
www.manaraa.com

power is the synchronous MMC PSA method, using either periodic or dynamic exchange schemes, whereby at fixed or computed intervals, all processes compare the current solutions, and move forward using the best solution from amongst all processes. This is similar to the synchronous MMC PSA method detailed in the next chapter. Finally, there is the asynchronous MMC PSA algorithm, whereby individual processes are able to communicate with a global state as needed. If the global state is superior to the current state on that process, then the global state is adopted. Otherwise, the global state is updated to reflect the new best solution. This scheme is similar to the asynchronous MMC PSA method detailed in the following chapter. One advantage of using multiple simultaneous Markov search chains is that even if a few individual chains become trapped in local minima, it is less likely that the entire solution will be trapped likewise. A further way to reduce the probability of becoming trapped in a local minimum and to increase the hill-climbing ability of the algorithm is to choose a new starting pattern randomly from all processes at each update step. In the Binary Branching scheme (described below), the new starting solution is selected by applying the Metropolis Criterion to successive, random pairings, much like a randomly seeded tournament bracket. A similar scheme, proposed by Chu [5] and applied to the multicycle nuclear fuel management problem by Kropaczek [17] is Parallel Simulated Annealing with Mixing of States. Unlike in the binary branching algorithm where a single starting point is used for all processes after an update step, the Mixing of States algorithm has each process start from a different point, chosen randomly from the set of previous Markov chain endpoints with a probability given by

$$p_i = \frac{\exp(-E_i/T_k)}{\sum_{j=1}^{n} \exp(-E_j/T_k)} \tag{1.7}$$

where $p_i$ is the probability of choosing the solution from process $i$, $E_i$ is the objective function value associated with solution $i$, and $T_k$ is the simulated annealing temperature of cooling step $k$.

Two of the PSA algorithms were chosen for implementation and testing in the `FORMOSA-B` program, namely the synchronous and binary-branching MMC PSA methods. Testing of the asynchronous algorithm is reserved for future work. In chapter 2, the `FORMOSA-B` implementation of each method is discussed, while in chapter 3 the results of a variety of numerical trials are used to compare these methods against each other and

للاستشارات
www.manaraa.com

against a serial benchmark solution. Finally, in chapter 4, several conclusions are drawn from this work and recommendations made for future work.

المنارة للاستشارات
www.manaraa.com

# Chapter 2

# MMCPSA

Multiple Markov Chain Parallel Simulated Annealing [19] (MMCPSA) is a parallelization scheme by which each process in the parallel operating environment (POE) follows an independent search path for a given interval. These intervals are punctuated by interactions with the other processes. The several variations of MMCPSA algorithms are characterized by the type of interprocess communication schemes they employ. Two particular MMCPSA schemes are here considered: Synchronous, where all processes communicate simultaneously, and Asynchronous, where each process is updated individually. The general pattern employed by all of these methods is depicted in figure 2.1.

## 2.1 Cooling Schedules

In parallelizing the `FORMOSA-B` code, the proven Huang cooling schedule [4] from the serial version was retained along with the optional fixed-decrement schedule. The adaptive cooling schedule computes the new simulated annealing temperature by multiplying the previous temperature by a factor of

$$\alpha = e^{-0.7T/\sigma} \tag{2.1}$$

where $T$ is the current temperature and $\sigma$ is the observed standard deviation in the augmented objective function during the previous cooling step. The standard deviation is calculated by keeping a running sum on each process of the objective function, the square of the objective function and the number of histories accepted. In synchronous simulated

المنارة للاستشارات
www.manaraa.com

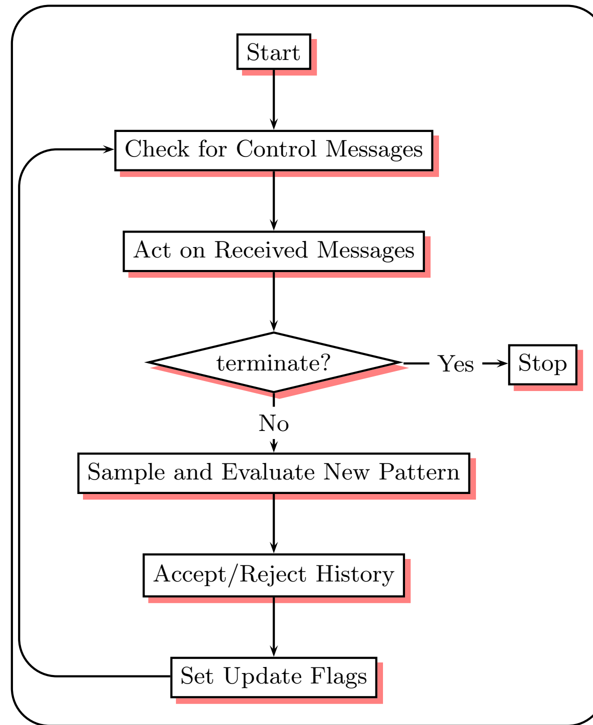


Figure 2.1: PSA Flowsheet

annealing the individual sums from each process are aggregated together to compute the overall standard deviation. In asynchronous annealing, however, the current global average state is not available to the process undergoing the update. For this situation, each time an update is performed with the root process, the partial sum for the latest search segment for the updating process is stored on the root process. An estimate of the global state is then made by summing over the set of the partial sums stored on the root process. This is explained further is section 2.3. Although this method does not give the exact value for the standard deviation of augmented objective function, it can serve as a useful estimate when the overall cooling rate is low.

## 2.2 Synchronous SA

Synchronous parallel simulated annealing splits the Markov search chain for each temperature interval between each of the $P$ processors in the parallel environment. Each

www.manaraa.com

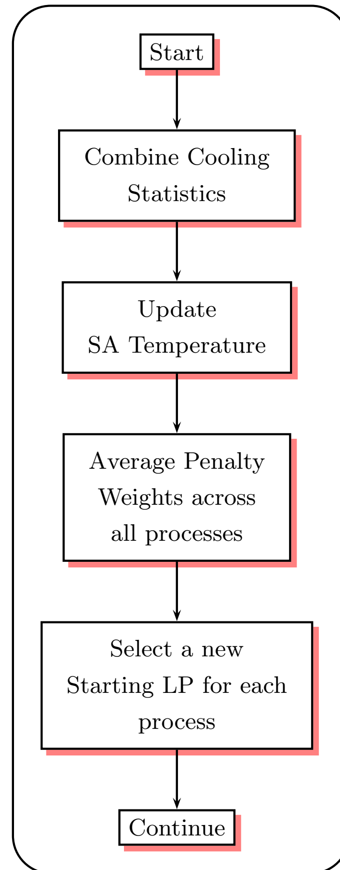


Figure 2.2: Global Update Logic

processor generates and accepts new loading patterns (LPs) and control rod programs (CRPs) at a constant annealing temperature until the root processor initiates a temperature update. These temperature update steps are initialized when more than a given fraction of the processes (7/8 was chosen for this study) have either a) sampled a given number, *lchain*, of histories or b) accepted a given number, *ltran*, of histories. When the cooling cycle initially begins, there are no statistics from which to compute an annealing temperature or penalty function weight values. Therefore an initial search of length *lsurv* is performed on all processors (see figure 2.3). During this search all non-grossly infeasible histories are accepted (i.e. an infinite annealing temperature is used). The statistics generated during this initial survey are then used to initialize the annealing temperature and penalty function weights. The choice of values for *lsurv*, *lchain* and *ltran* will be covered in more detail in

www.manaraa.com

chapter 3.

The three key characteristics of the synchronous mode of MMCPSA are: 1) all processes use the same SA temperature, 2) all processes use the same penalty multipliers, and 3) all processes adjust the temperature and penalty multipliers at the same time. Traits 1 and 2 are assured because they are based on the cooling statistics, and the statistics for the previous segment are combined across all processes concurrently (see figure 2.2).

In addition to determining the annealing temperature and penalty function weights, each update step must also determine the choice of the next starting configuration. There are several different methods available for choosing this configuration, each of which seeks to find a balance between the opposing requirements of minimization and solution diversity. One simple strategy is to compare the best solution accepted by each processor during the search and choose that with the lowest augmented objective function value. As implemented in `FORMOSA-B`, this method is referred to as the *Synchronous-Best-Solution* mode of operation (sync, for short). Increasing the complexity and solution diversity by one step is the *Stochastic Tournament* method. In this method, the best solutions from each process are first randomly paired, then downselected using the Metropolis criterion. The process is repeated for the surviving solutions until a single configuration remains. This configuration is then carried forward as the starting point for every process. Because this method selects a solution by repeatedly pairing and eliminating candidates it is referred to as the Binary Branching mode of operation. A third method, not currently implemented in `FORMOSA-B`, is the Mixing-of-States algorithm. In this method each process would select its own starting point from the set of best solutions according to equation 1.7.

One additional feature utilized by both the synchronous and asynchronous modes is the *Return-to-base* strategy for recovering from an infeasible search space. If, through the course of the optimization, a processor samples more than $lchain/4$ consecutive grossly infeasible histories, it will send a message to the root process indicating that it is stuck. The root process then transmits the best solution yet found, and the stuck node resumes from this configuration (see figure 2.4 below).

المنارة للاستشارات
www.manaraa.com

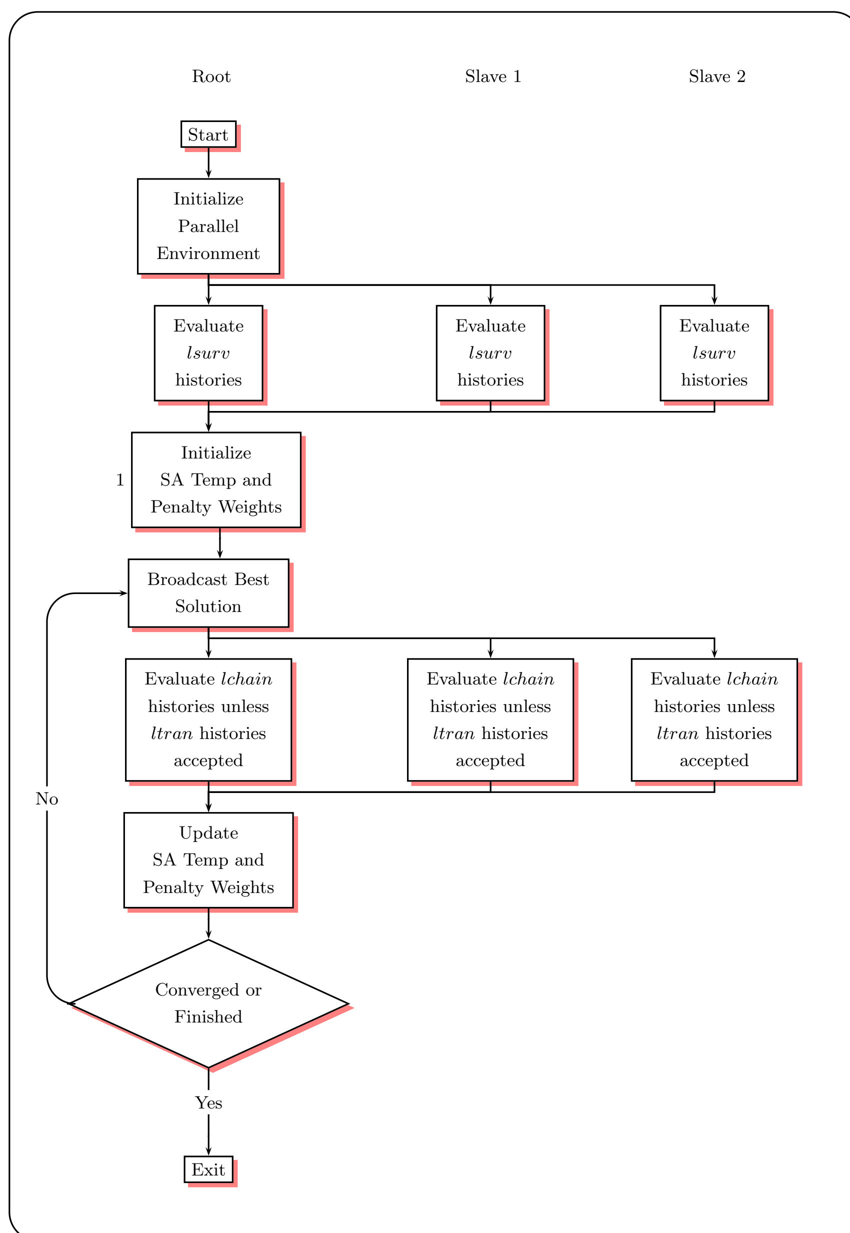


Figure 2.3: Synchronous Parallel Simulated Annealing Program Flow

المنارة للاستشارات
www.manaraa.com

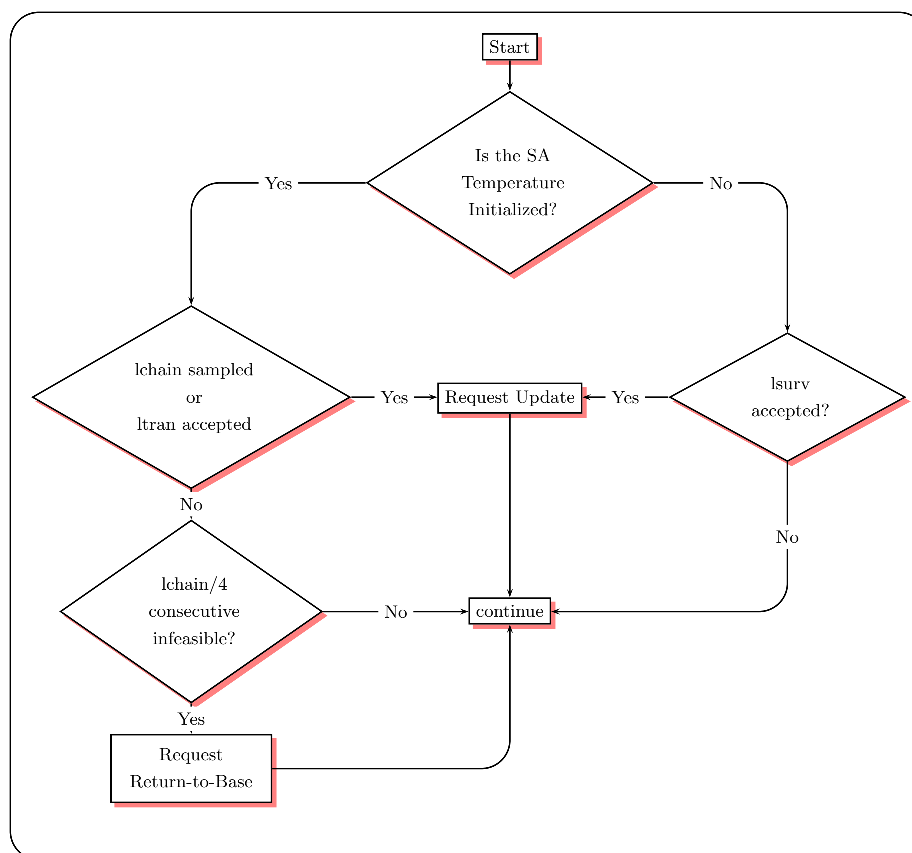


Figure 2.4: Set Update Flags

www.manaraa.com

## 2.3 Asynchronous SA

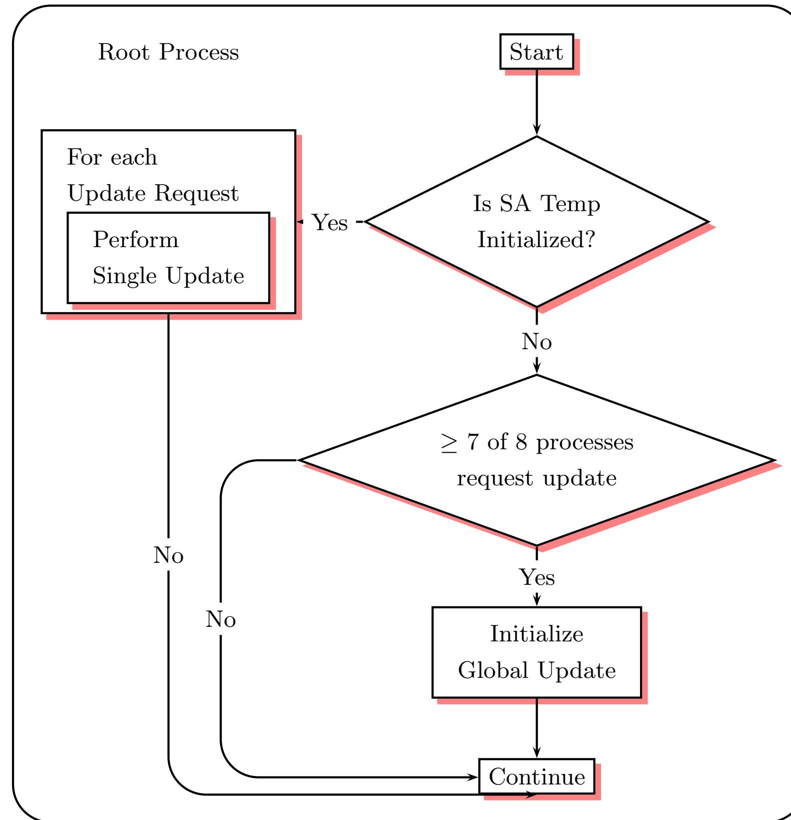


Figure 2.5: Asynchronous Update Request Handling

The asynchronous simulated annealing mode has two aims. First is to reduce the burden of interprocess communication by minimizing collective communications that require coordination amongst all of the processes. The second is to accelerate the cooling rate in order to hasten the completion of the optimization cycle. Asynchronous SA differs from synchronous SA in that after the initial temperature survey, each process is able to perform a temperature update step independently of any other process. After sampling *lchain* valid histories or accepting *ltran* transitions, the process sends a signal to the root process requesting an annealing temperature update. It then continues sampling until the root process sends a return signal initiating the update step (see figure 2.5). At the start of the update step, the process transmits the cooling statistics from the previous

المنارة للاستشارات
www.manaraa.com

segment to the root process, where they are stored. The root process then sends back the average cooling statistics for all processors, which are then used to calculate the new annealing temperature and penalty function weights. The updating process then transmits the unaugmented objective function and constraint violation values of its best solution to the root, and vice versa. These values are then used, along with the local penalty weights, to compute an augmented objective function value. On the root process, if the new augmented objective value is lower than the current minimum value, then the corresponding loading pattern is installed on the root as the global best solution. The augmented objective function is similarly reconstructed for the global best solution on the updating process; however, the global best solution is accepted over the local best solution by use of the Metropolis Criterion.

## 2.4 Interprocess Coordination and Communication

Interprocess communication is structured around a radial topology whereby all communication is directed to or from the so-called *root process*. The root process is simply that process which is assigned a rank of zero by the parallel operating environment. The nonblocking routine `MPI_IRECV` is used to create a communication channel between the root process and each other process in the parallel environment. This communication routine reserves a given variables as a message buffer. The `MPI_TEST` routine can then be used to test for the presence of a received message. Thus, each process is always listening for messages from the root process. The received messages are handled differently on the root process than on the slave processes. The root process first checks for messages from all other processes, then computes whether an update is required, and if so, which type. If an update is to be initiated, it sends out control messages to all participating processors. When the slave processes again reach the top of the simulated annealing loop, they first check for received messages, and if any are found, they are immediately acted upon. The longest period that the root process would have to wait for a response from a slave process would be the length of time required for a Control-Rod-Programming Update step. Two advantages of this coordination scheme are that 1) it allows the root process to execute its own Markov search chain, rather than sitting idle, waiting for the next update step and 2)

www.manaraa.com

it does not require implementing complex multi-threaded code or interrupts. In addition to control messages, several MPI routines are used to communicate data between processes. Because data communications occur in a deterministic fashion, they can be implemented using simple blocking communications routines.

المنارة للاستشارات
www.manaraa.com

# Chapter 3

# Numerical Results

The driving purpose of this work is to demonstrably shorten the FORMOSA-B runtimes in an efficient manner. A true performance assessment encompasses two related ideas: robustness and efficiency. Robustness is taken as the ability of the program to avoid becoming trapped in a locally feasible solution. Efficiency relates the way in which the runtimes scale with processor count for a given quality of solution.

## 3.1 Testing Environment

Computational testing of both the serial and parallel FORMOSA-B codes was performed using the Henry2 cluster located in the High Performance Computing Center at North Carolina State University [27]. This cluster consists of a mixture of 612 individual IBM BladeCenter [31] compute nodes, each with dual Intel Xeon processors (in single-, dual- and quad-core varieties). Furthermore, each node has 2 gigabytes of memory per core with each node interconnected via two Gigabit Ethernet links. FORMOSA-B is coded in FORTRAN and compiled with version 8.1 of the Intel Fortran Compiler [6]. Parallel job scheduling is handled by the Platform LSF load-manager software. This software automatically controls the allocation and assignment of the computational nodes to different users or projects based upon a specified priority queue structure. For this work, a total of 92 processors were available on a priority basis, with parallel runs generally starting within 30 seconds of their being submitted to the queue.

المنارة للاستشارات
www.manaraa.com

## 3.2 Test Cases

In order to test the performance of the parallel simulated annealing algorithm a quarter-core model of a 368-assembly GE-4 BWR reload core was utilized. An example of an optimized loading pattern is shown in table 3.1; note that the fuel assembly numbers reference the ID numbers assigned to each assembly type within the `FORMOSA-B` input deck and do not by themselves indicate any particular properties of the fuel.

Table 3.1: 368-Assembly Quarter-Core Loading Pattern

| $C_L$ | 23 | 25 | 27 | 29 | 31 | 33 | 35 | 37 | 39 | 41 | 43 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 22 | 6 | 64 | 45 | 65 | 23 | 64 | 47 | 64 | 44 | 61 | 30 |
| 20 | 64 | 29 | 52 | 16 | 63 | 24 | 63 | 63 | 27 | 42 | 9 |
| 18 | 49 | 51 | 31 | 65 | 53 | 64 | 48 | 63 | 37 | 8 | 12 |
| 16 | 65 | 4 | 65 | 59 | 64 | 60 | 64 | 63 | 28 | 19 | |
| 14 | 5 | 63 | 46 | 64 | 38 | 64 | 35 | 62 | 2 | | |
| 12 | 64 | 7 | 64 | 58 | 64 | 11 | 55 | 39 | 13 | | |
| 10 | 54 | 63 | 50 | 64 | 62 | 57 | 40 | 21 | 32 | | |
| 8 | 64 | 63 | 63 | 63 | 64 | 36 | 10 | 1 | | | |
| 6 | 41 | 25 | 43 | 26 | 22 | 15 | 20 | | | | |
| 4 | 56 | 34 | 3 | 14 | | | | | | | |
| 2 | 33 | 17 | 18 | | | | | | | | |

This model simulates a core running with a thermal power of 1911.6MW through a cycle burnup of 12,336 MWD/MTU with conventional-core control rod programing (COC).

المنارة للاستشارات
www.manaraa.com

The cycle is discretized into 17 timestep increments with 1,250 MWD/MTU of burnup accrued per time-step (two time-steps are used when rod swaps occur.) Many key optimization settings were held fixed throughout the tests that follow. A summary of these values is presented below in table 3.2.

Table 3.2: Optimization Test Case Settings

**Thermal Limits**

| | |
|---|---|
| MFLPD | 0.94 |
| MAPRAT | 0.94 |
| MFLCPR | 0.96 |

**Reactivity Limits**

| | |
|---|---|
| Minimum Shutdown Margin | $0.025\Delta k/k$ |
| BOC Hot Excess Reactivity Limit | $1820pcm$ |
| Maximum Allowed Reactivity Shuffle | $0.15\Delta k$ |

**Burnup Limits**

| | |
|---|---|
| Pin | 65,000 MWD/MTU |
| Assembly | 50,000 MWD/MTU |
| Region | 45,000 MWD/MTU |

**Symmetry Constraint**

| |
|---|
| Quarter Core Reflective Loading |
| Eighth Core Fuel Exchanges |

**Miscellaneous Settings**

| | |
|---|---|
| Optimization Objective | EOC Flow Minimization |
| CRP Heuristic Objective | Flow Minimization |
| Critical Flow Search | Enabled |
| Number of F/A Mechanical Types | 2 |
| Shutdown Margin Calculation | All Interior Control Rods |
| Assemblies available in Fuel Pool | 13 |
| Pin Power Reconstruction on the most reactive 20% of nodes. | |

## 3.3 Performance Metrics

The ultimate goal and purpose for parallelizing the FORMOSA-B program is to generate an equal or superior set of solutions with a greatly reduced turnaround time. It is a

المنارة للاستشارات
www.manaraa.com

simple matter to compute and compare turnaround times on a program. It is much more difficult, however, to evaluate the performance of an asymptotically convergent stochastic algorithm executing in a heterogeneous computational environment. The objective function and constraint violation values computed during the optimization run vary as a function of the randomly generated loading pattern, the adaptively computed penalty constraint multipliers, the optimization settings and the reactor model itself; a simple comparison of average objective function versus number of samples will not suffice to compare different optimization trials. Instead, as a basis of comparison, two factors are considered. First, does the parallel run produce optimized solutions of similar quality to those generated during the serial run? This is established by comparing the distributions of the objective function and constraint violation values for the best solution (i.e. the solution archived with the lowest objective function value) found by a repeated sampling of individual runs.

Once it has been established that the runs are converging to similar optimized states, parallel speedup is assessed by comparing the rate at which new solutions are sampled. Speedup is calculated via the following equation:

$$S = \frac{T_0/N_0}{T/N} \tag{3.1}$$

where $N$ and $N_0$ are the number of histories specified or generated during a given trial and reference case, respectively; $T$ and $T_0$ are likewise the execution time for the compared cases. It is desired that the parallel speedup and efficiency calculations reflect the performance of the PSA algorithm itself while compensating for the effects of variations in the underlying computational hardware. This is particularly necessary given the heterogeneous nature of the Henry2 cluster, where processing nodes of various vintages and speeds may be encountered both from one run to another and within a single parallel run. This is done by introducing a normalization factor for each run. This factor is given by

$$F = \frac{N_{LP}T_{LP} + N_{CRP}T_{CRP}}{N_{LP} + N_{CRP}} \tag{3.2}$$

where $N_{LP}$ is the total number of loading patterns sampled during the given run and $T_{LP}$ is the mean time required per loading pattern evaluation (likewise for $N_{CRP}$ and $T_{CRP}$). Note that LP and CRP evaluation times are automatically calculated and recorded by the code each time either subroutine is called. Thus, the compensated speedup value $\tilde{S}$ is given

www.manaraa.com

by the equation:

$$\tilde{S} = \frac{F}{F_0} \times S \tag{3.3}$$

where the value of $F$ is derived from the parallel optimization case under consideration and $F_0$ is computed from the serial benchmark data. This factor provides a rough correction for the effects of variable processor power, but it does not compensate for imbalances that may occur when a subset of the processors in the parallel environment are significantly faster or slower than the others. In these situations parallel speedup and efficiency can be expected to suffer. Further, the variation of the ratio of CRP updates to LP updates between a given trial and reference case will degrade the quality of the correction.

## 3.4 Benchmarks

To establish a baseline for comparison, three serial `FORMOSA-B` runs were performed using the default run settings (*lchain*, *ltran*, etc.) and different random number seeds. These runs provide a benchmark for comparing both the speedup achieved by the parallel algorithms as well as the convergence properties and final solution quality.

Table 3.3: Serial Benchmark Results

**Settings**

| *llngth* | *lchain* | *ltran* | *lsurv* | *ilimx* |
|---|---|---|---|---|
| 16,000 | 800 | 320 | 800 | 100 |

**Best Solution Results**

| | # Sampled | # Accepted | Min. OF | Run Time |
|---|---|---|---|---|
| Run 1 | 16,194 | 4,060 | -0.1145 | 37.1hr |
| Run 2 | 16,694 | 3,430 | -0.1019 | 52.0hr |
| Run 3 | 16,286 | 4,436 | -0.1015 | 45.2hr |

**Best Solution Constraint Violations (x1,000,000)**

| | MFLPD | MAPRAT | MFLCPR | CSDM | Max. HX | Flow |
|---|---|---|---|---|---|---|
| Run 1 | 72.8 | 10.4 | 0.0 | 0.0 | 0.0 | 2,166.5 |
| Run 2 | 115.1 | 0.0 | 0.0 | 113.2 | 0.0 | 11,777.5 |
| Run 3 | 62.8 | 0.0 | 0.0 | 0.0 | 0.0 | 1,398.8 |

Several metrics reported in the above table 3.3 (and again later) merit further explanation. First, the precise meaning of *llngth*, *lchain*, *ltran*, *lsurv* and *ilimx* are as

www.manaraa.com

follows:

- **llngth:** This is the target number of histories to be sampled during the simulated annealing cooling cycle. The actual number of histories sampled will be greater, as the program only evaluates this termination criteria during update steps.

- **lchain:** This is the maximum number of histories that a single process will evaluate before requesting the next temperature update step.

- **ltran:** This is the maximum number of histories that a single process will accept before requesting the next temperature update step.

- **lsurv:** This is the number of histories that each process will evaluate during the temperature initialization phase of the cooling cycle.

- **ilimx:** This parameter determines the frequency at which the Control Rod Program is optimized via built-in heuristic rules. For example an *ilimx* value of 200 indicates that for every 200 loading patterns evaluated, the CRP is optimized once.

Due to the stochastic nature of the SA algorithm, it is not expected that repeated optimization runs will find identical optimized loading patterns. Instead, optimized solutions should lie along a tradeoff surface given by the objective function and penalty constraint violations. Furthermore, due to the adaptive penalty constraint multiplier algorithm, optimization runs that specify different *llngth* values are not expected to converge at the same rate. Thus in order to compare the computational performance of the parallel algorithm, certain assumptions and simplifications are made. Rather than computing a rate or order of convergence, as would be typical for most deterministic computations, speedup calculations are based on the average rate at which histories are sampled over the cooling cycle. In order to confirm that the optimized solutions are indeed optimal, they are compared to the base case results in terms of final objective function (i.e. EOC flow reduction) and remaining constraint violation values. Note that no attempt is yet made to correlate optimized solution quality to the total number of sampled solutions. Also note that as the term "constraint violation" indicates, one goal of the optimization is the removal of all constraint violations. However, it is not always possible for a given set of constraints to find a solution that has no violations.

المنارة للاستشارات
www.manaraa.com

To initially establish the performance of the parallel algorithm, three repeated trials were run with each of three optimization settings. Parallel runs were performed on four processors in the synchronous best-solution mode with CRP optimization occurring prior to each temperature update. Results are presented below in table 3.4.

Table 3.4: Benchmark Comparison Results

| # Processors | 1 | 4 | 4 |
|---|---|---|---|
| # Samples Requested | 16,000 | 16,000 | 24,000 |
| % EOC Flow Change | -10.6± 0.74% | -10.2± 0.1% | -10.6± 0.64% |
| Best Solution Constraint Violations (x1,000,000) | | | |
| MFLPD | 83.6± 27.8 | 76.4± 97.1 | 70.8± 141.5 |
| MAPRAT | 3.5± 6.0 | 0.0± 0.0 | 4.4± 5.4 |
| MFLCPR | 0.0± 0.0 | 24.3± 48.6 | 137.0± 272.4 |
| CSDM | 37.7± 65.3 | 0.0± 0.0 | 0.7± 1.3 |
| Max. HX | 0.0± 0.0 | 0.0± 0.0 | 0.0± 0.0 |
| Crit. Flow | 5114.3±5783.3 | 1872.1±752.9 | 4615.7±3225.6 |

Both 16,000 and 24,000 sample optimization runs exhibit similar EOC flow reduction and constraint violation values. Two conclusions can be inferred from this: first, both parallel and serial algorithms are converging to similar solutions (the parallel algorithm works). Second, increasing the number of samples from 16,000 to 24,000 does not result in a markedly better best solution; this indicates that for the reactor model under consideration, an increase of sample size above 16,000 does not greatly improve the optimization results. This result is likely a factor of the smaller optimization search space of the chosen reactor model.

## 3.5 Tuning the Parallel Algorithm

The chosen parallel algorithms include a number of search parameters, the choice of which will affect the quality and robustness of the cooling cycle. Two parameters of particular interest are the CRP Update Frequency and the minimum Markov Chain segment length.

www.manaraa.com

### 3.5.1 CRP Update Frequency

As the loading pattern (LP) evolves, so does the optimum control rod program (CRP) [11]. By updating the CRP to reflect the changes in the reactivity and power distribution the program is able to more effectively reduce or eliminate constraint violations. However, the heuristic CRP optimization routine requires a factor of 10 to 15 greater computation time than a simple LP evaluation. This, combined with the observation that the optimized CRPs vary only slightly for similar loading patterns indicates that CRP optimization is not required after each an every LP shuffle. In the serial algorithm, CRP updates are automatically performed after a user-determined number of loading pattern shuffles have been evaluated. The parallel algorithm additionally performs a CRP optimization on the best solution found by each segment immediately before this solution is used in the temperature update step. However, if the Markov Chain is short between temperature updates, then it may not even be necessary to reoptimize the CRP prior to the update. The decision of whether to require CRP optimization prior to temperature update is thus based on whether the total number of LPs sampled across all processes exceeds a user-specified CRP Update Threshold value.

A series of test runs were performed using four processors for a range of CRP Update Threshold values between 20 and 1,000 with a nominal Markov Chain size of 800 samples or 400 acceptances (*lchain = 200*, and *ltran = 100* ) with the CRP reoptimized once for every 250 LP samples evaluated within each Markov Chain segment. Parameters and results are presented in tables 3.5 and 3.6 and figures 3.1 through 3.7.

This series of tests confirms the expectation that using loading pattern optimization without simultaneously optimizing the control rod program results in faster execution times at the cost of greatly increased penalty constraint violations. The only constraint violation not affected by the absence of CRP optimization is the cold-shutdown margin, which is largely independent of the operating control rod pattern. The range of CRP Update Threshold values tested ranged from well below the nominal Markov chain length to slightly above it. The optimization results and speedup values do not seem to vary significantly as a result of changing CRP Update Threshold. It is likely that the choice of an optimum CRP update frequency is closely related to the particular reactor model under consideration; further testing is required to elucidate any such trends before definitive

المنارة للاستشارات
www.manaraa.com

Table 3.5: CRP Speedup Results

| CRP Threshold | # of Runs | Avg. # of Samples | Average Runtime[hr] | End-of-Cycle % Flow Change | Speedup | Efficiency |
|---|---|---|---|---|---|---|
| Serial | 3 | 16,391 | 44.8 | -10.60±0.74% | - | - |
| -[1] | 5 | 24,349 | 17.6 | -9.21±3.12% | 3.43 | 85.7% |
| 1,000 | 5 | 24,518 | 21.2 | -10.18±0.18% | 3,21 | 80.2% |
| 700 | 5 | 24,474 | 18.2 | -10.35±0.40% | 3.22 | 80.6% |
| 400 | 5 | 24,561 | 21.9 | -8.57±4.23% | 3.21 | 80.3% |
| 200 | 5 | 24,451 | 20.9 | -10.33±0.61% | 3.20 | 80.0% |
| 140 | 4 | 25,155 | 22.5 | -10.16±0.10% | 3.22 | 80.4% |
| 100 | 4 | 24,571 | 20.4 | -10.16±0.10% | 3.21 | 80.1% |
| 80 | 4 | 24,598 | 21.2 | -10.39±0.39% | 3.23 | 80.7% |
| 60 | 5 | 24,341 | 21.8 | -10.09±0.09% | 3.21 | 80.4% |
| 40 | 3 | 24,444 | 21.8 | -10.13±0.21% | 3.21 | 80.4% |
| 20 | 4 | 24,397 | 21.1 | -10.57±0.64% | 3.23 | 80.8% |

1: CRP Optimization Disabled

conclusions can be drawn.

www.manaraa.com

Table 3.6: CRP Optimization Results

| CRP Threshold | Constraint Violation (x1,000,000) MFLPD | MAPRAT | MFLCPR |
|---|---|---|---|
| Reference | 0.00± - | 224.93 ± - | 0.00± - |
| Serial | 83.56± 27.75 | 3.45 ± 5.98 | 0.00± 0.00 |
| -[1] | 301.21± 178.04 | 77.41 ± 72.60 | 820.87± 627.07 |
| 1,000 | 129.65± 156.45 | 11.25 ± 13.61 | 34.15± 59.24 |
| 700 | 119.78± 159.46 | 12.91 ± 18.30 | 148.22± 293.40 |
| 400 | 174.14± 77.79 | 19.20 ± 42.92 | 83.11± 185.85 |
| 200 | 188.63± 116.09 | 22.00 ± 49.19 | 454.28± 762.19 |
| 140 | 159.41± 191.17 | 56.53 ±113.07 | 417.57± 497.16 |
| 100 | 122.22± 83.64 | 17.06 ± 23.34 | 161.11± 322.22 |
| 80 | 151.11± 127.38 | 31.50 ± 63.01 | 47.76± 59.87 |
| 60 | 106.27± 121.02 | 0.24 ± 0.55 | 0.00± 0.00 |
| 40 | 153.21± 164.60 | 0.00 ± 0.00 | 0.00± 0.00 |
| 20 | 70.76± 141.53 | 4.44 ± 5.38 | 137.02± 272.36 |

| CRP Threshold | Constraint Violation (x1,000,000) CSDM | MAXHX | Critical Flow |
|---|---|---|---|
| Reference | 0.00± - | 0.00 ± - | 0.34± - |
| Serial | 37.72± 65.34 | 0.00 ± 0.00 | 5,114.26± 5,783.27 |
| -[1] | 146.24± 236.34 | 52.66 ±117.75 | 21,297.52± 4,326.00 |
| 1,000 | 426.70± 889.00 | 19.71 ± 44.07 | 2,253.01± 476.36 |
| 700 | 213.78± 314.65 | 0.00 ± 0.00 | 5,516.08± 4,840.76 |
| 400 | 118.68± 174.31 | 160.98 ±324.52 | 6,850.61± 3,359.02 |
| 200 | 357.71± 724.19 | 109.09 ±236.20 | 8,682.35± 7,209.18 |
| 140 | 621.98±1,234.86 | 0.00 ± 0.00 | 12,164.24±12,566.13 |
| 100 | 30.18± 60.37 | 265.80 ±531.61 | 3,738.07± 3,077.43 |
| 80 | 44.78± 89.57 | 0.00 ± 0.00 | 6,980.28± 7,055.19 |
| 60 | 122.90± 274.82 | 0.00 ± 0.00 | 3,586.27± 2,254.69 |
| 40 | 0.00± 0.00 | 0.00 ± 0.00 | 2,357.21± 664.53 |
| 20 | 0.66± 1.32 | 0.00 ± 0.00 | 4,615.66± 3,225.62 |

1: CRP Optimization Disabled

المنارة للاستشارات
www.manaraa.com

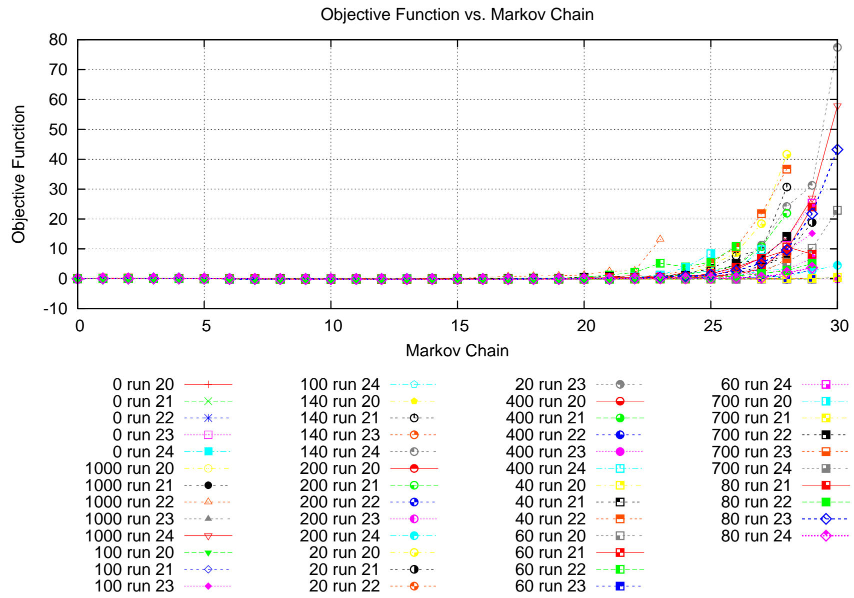


Figure 3.1: CRP Markov-Chain Averaged Objective Function

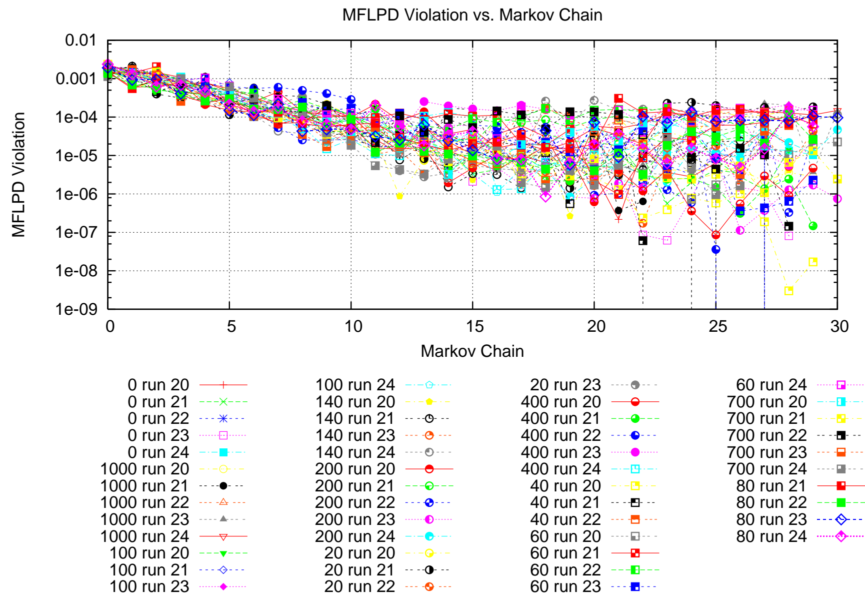


Figure 3.2: CRP Markov-Chain Averaged MFLPD

المنارة للاستشارات
www.manaraa.com

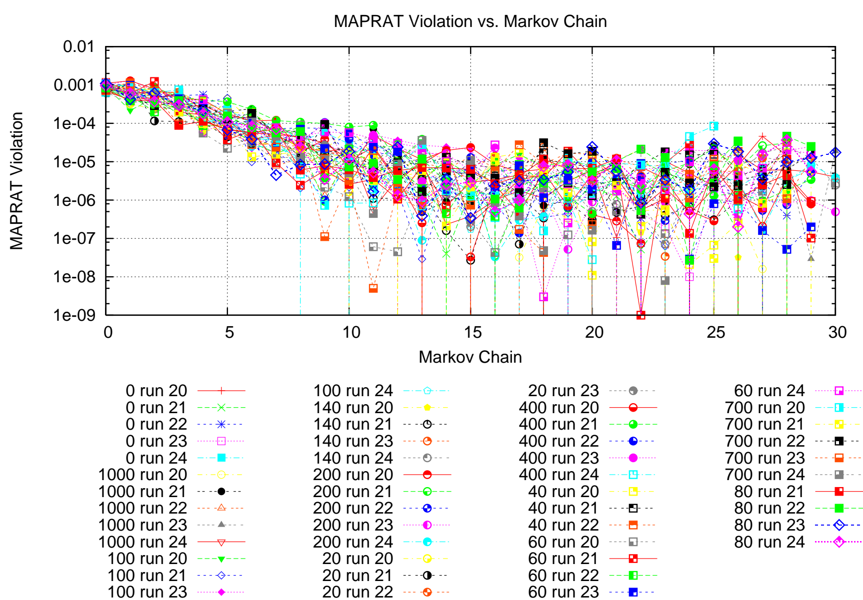


Figure 3.3: CRP Markov-Chain Averaged MAPRAT

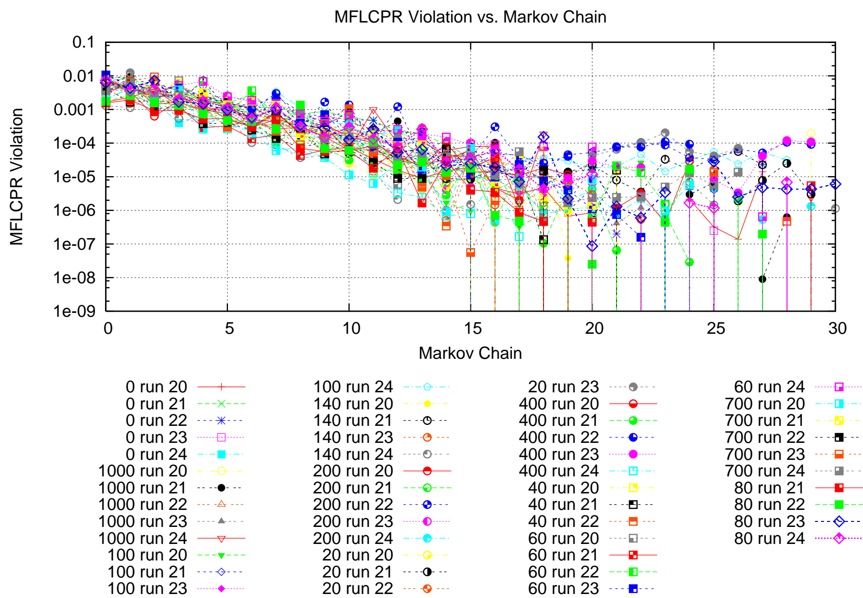


Figure 3.4: CRP Markov-Chain Averaged MFLCPR

المنارة للاستشارات
www.manaraa.com

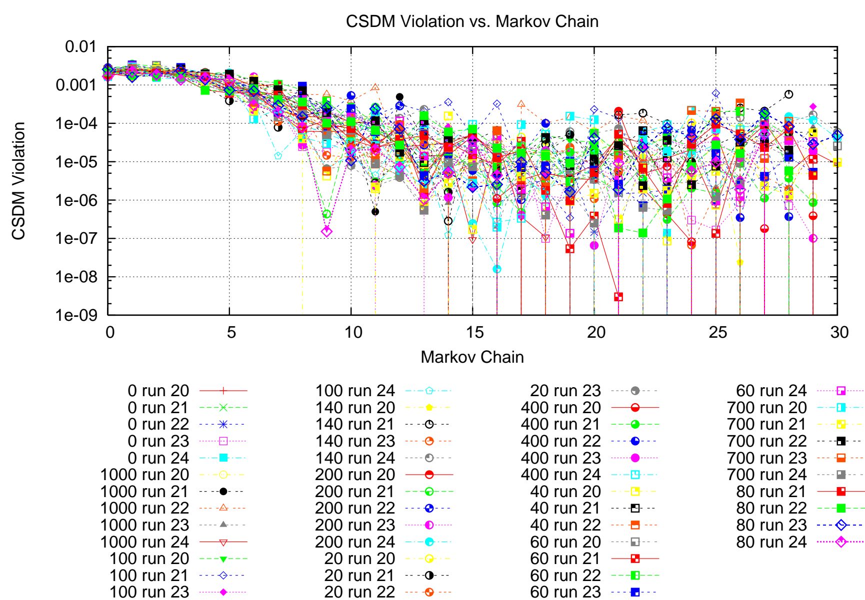


Figure 3.5: CRP Markov-Chain Averaged CSDM

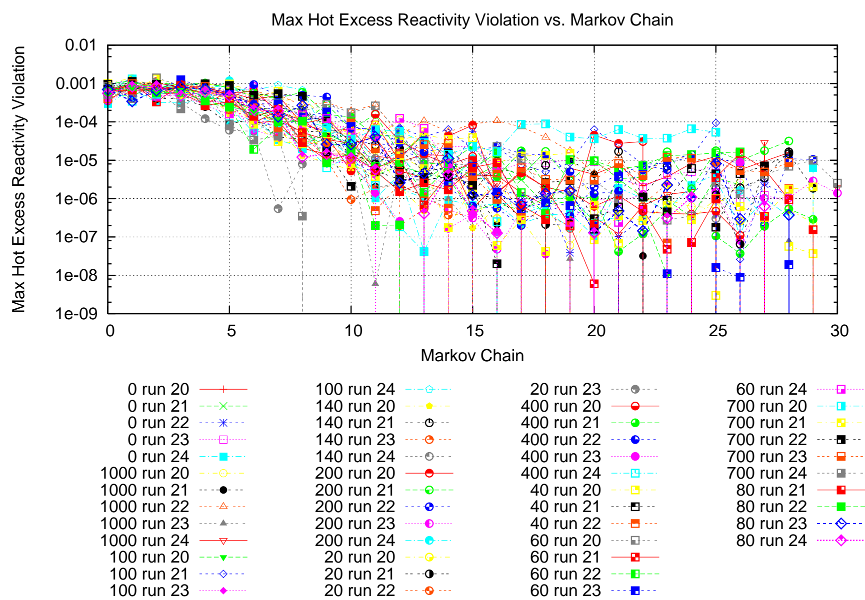


Figure 3.6: CRP Markov-Chain Averaged Max HX

www.manaraa.com

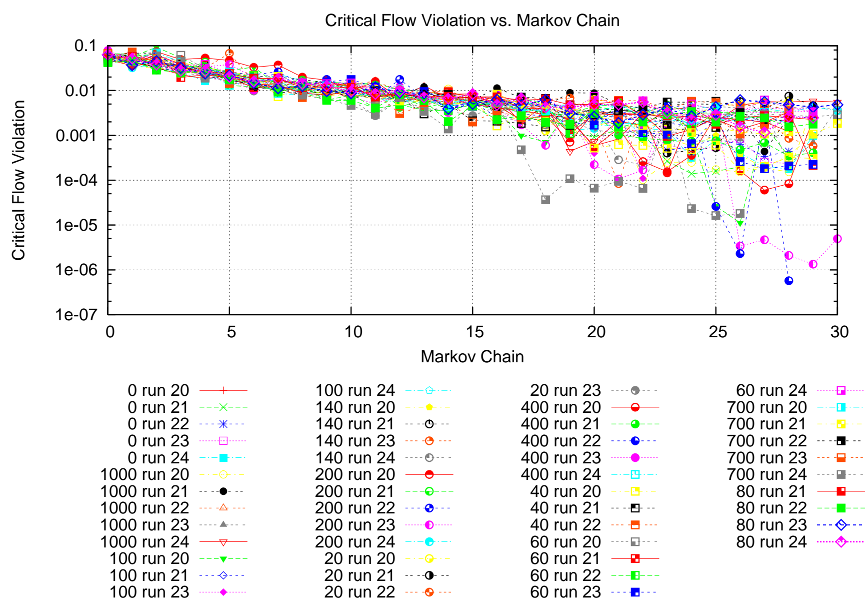


Figure 3.7: CRP Markov-Chain Averaged Flow Violation

المنارة للاستشارات

www.manaraa.com

### 3.5.2 Depth versus Width

In parallelizing the Markov search chain, a tradeoff must be made. By splitting the search into many short pieces great reductions in execution time may be obtained. However, these gains occur at the expense of the depth and extent of the search chain. In order to move between any two points in the configuration space, a certain minimum number of individual moves or exchanges must be made. The probability of successfully making this transition is a function of both the energy-landscape of the intervening configurations and the number of moves available. The greater the number of transitions attempted, the greater the odds that the sequence will pass through. The minimum Markov chain segment length must be chosen so as to allow an adequate coverage of the search space. In order to test where these limits may lie, a series of cases were generated whereby a single Markov search chain of a given length was divided between several sets of processors.

This set of test cases utilize a long, 1,200 history Markov chain size and a 48,000 history requested search size. These higher values were chosen to ensure the Markov Chain segments are fully equilibrated before temperature decrements and to ensure that the parallel sample size is large enough that trends can be separated from other stochastic effects. Again the Markov chain is divided amongst 2, 4, 8, 16, 20 and 48 processors. Optimization settings are detailed below in table 3.7 with results following in tables 3.8 and 3.9 and figures 3.8 through 3.14.

Table 3.7: Depth versus Width Settings

| # proc. | *llngth* | *lchain* | *ltran* | *lsurv* | CRP Update Threshold |
|---|---|---|---|---|---|
| 2 | 48,000 | 600 | 300 | 2500 | 1,000 |
| 4 | 48,000 | 300 | 150 | 1250 | 1,000 |
| 8 | 48,000 | 150 | 75 | 625 | 1,000 |
| 16 | 48,000 | 75 | 38 | 313 | 1,000 |
| 20 | 48,000 | 60 | 30 | 250 | 1,000 |
| 48 | 48,000 | 60 | 30 | 150 | 1,000 |

Several observations can be drawn from these results. The first is that although the 48-processor runs had optimization parameters similar to the 20-processor runs, it exhibits higher levels of and variability in constraint violations. This is likely caused by the fact that the chain segment lengths and total sample size were the same, requiring the 48-processor

www.manaraa.com

Table 3.8: Depth versus Width Speedup Results

| # of Processes | # of Runs | Avg. # of Samples | Average Runtime[hr] | End-of-Cycle % Flow Change | Speedup | Efficiency |
|---|---|---|---|---|---|---|
| 2 | 2 | 48,244 | 69.3 | -10.1± 0.1% | 1.92 | 95.8% |
| 4 | 3 | 49,128 | 38.1 | -10.2± 0.3% | 3.52 | 88.1% |
| 8 | 3 | 48,978 | 20.4 | -10.1± 0.1% | 6.59 | 82.4% |
| 16 | 3 | 48,731 | 11.8 | -10.0± 0.0% | 11.30 | 70.6% |
| 20 | 3 | 48,759 | 9.1 | -10.1± 0.1% | 14.63 | 73.1% |
| 48 | 2 | 50,303 | 4.2 | -10.1± 0.1% | 32.72 | 68.2% |

Table 3.9: Depth versus Width Optimization Results

| # of Processes | Constraint Violation (x1,000,000) | | |
|---|---|---|---|
| | MFLPD | MAPRAT | MFLCPR |
| 2 | 273.01± 386.09 | 44.92 ± 63.53 | 0.00± 0.00 |
| 4 | 177.85± 155.08 | 12.39 ± 21.47 | 101.62± 139.44 |
| 8 | 277.12± 226.79 | 0.00 ± 0.00 | 385.51± 667.73 |
| 16 | 176.05± 177.26 | 26.26 ± 45.48 | 378.04± 395.53 |
| 20 | 194.16± 42.11 | 6.98 ± 9.55 | 33.00± 57.16 |
| 48 | 295.40± 380.57 | 131.62 ±186.14 | 1,241.59± 1,728.21 |

| CRP Threshold | Constraint Violation (x1,000,000) | | |
|---|---|---|---|
| | CSDM | MAXHX | Critical Flow |
| 2 | 144.73± 204.68 | 70.41 ± 99.57 | 6,613.26± 437.90 |
| 4 | 26.44± 45.79 | 0.00 ± 0.00 | 4,253.82± 2,136.01 |
| 8 | 551.40± 932.22 | 83.29 ±144.27 | 13,844.71±17,802.49 |
| 16 | 659.94± 594.81 | 336.94 ±304.07 | 18,980.40±11,137.66 |
| 20 | 638.55±1,105.99 | 307.76 ±533.05 | 7,843.45± 9,882.08 |
| 48 | 1,200.83±1,340.64 | 569.04 ±804.74 | 22,118.72±24,318.19 |

المنارة للاستشارات
www.manaraa.com

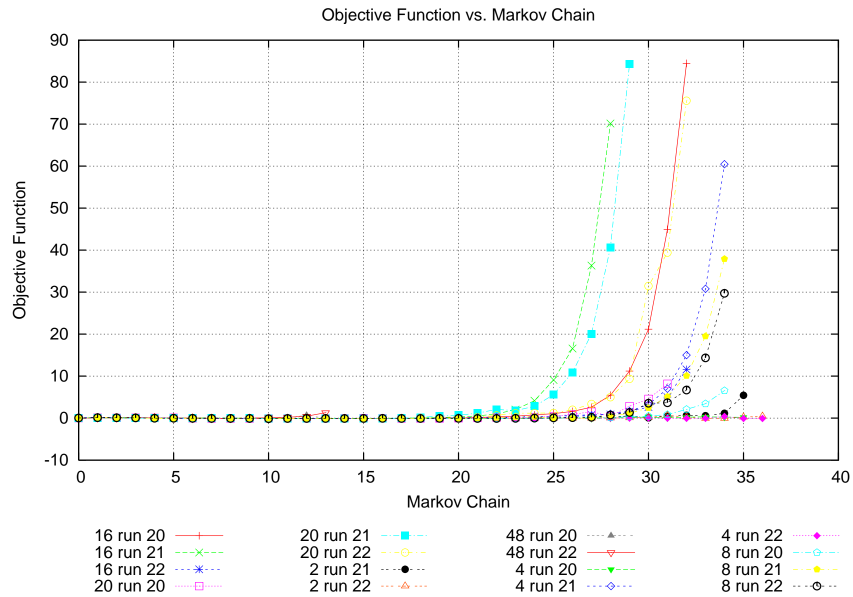


Figure 3.8: Depth versus Width Markov Chain Averaged Objective Function

case to utilize fewer cooling steps. Further, it can be seen that parallel efficiency declines as the number of processors increases. This result is expected as greater parallelization limits the search coverage and increases communication overhead.

المنارة للاستشارات
www.manaraa.com

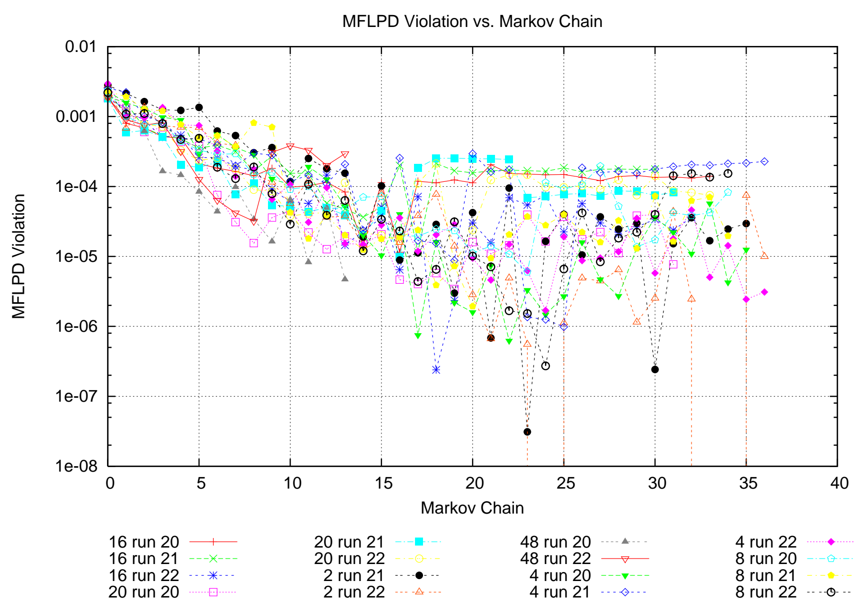


Figure 3.9: Depth versus Width Markov Chain Averaged MLFPD

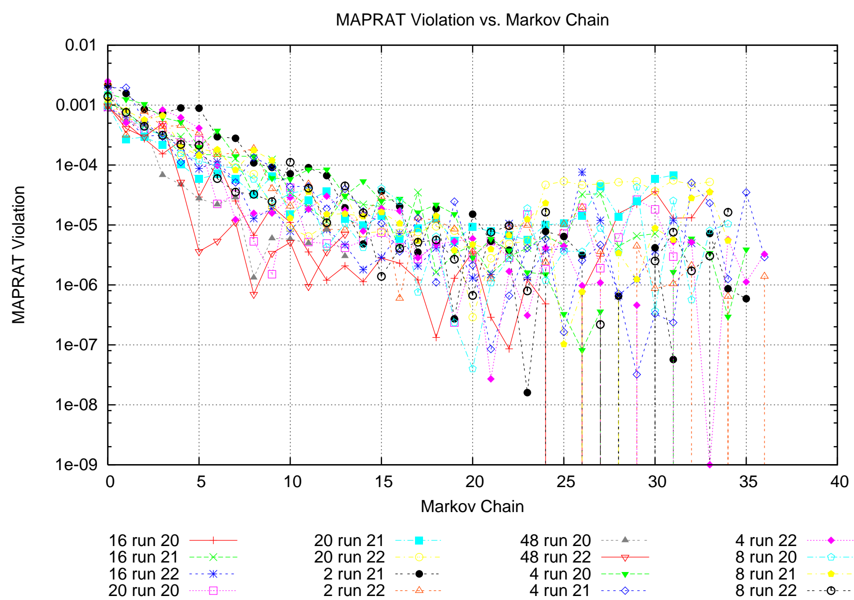


Figure 3.10: Depth versus Width Markov Chain Averaged MAPRAT

المنارة للاستشارات
www.manaraa.com

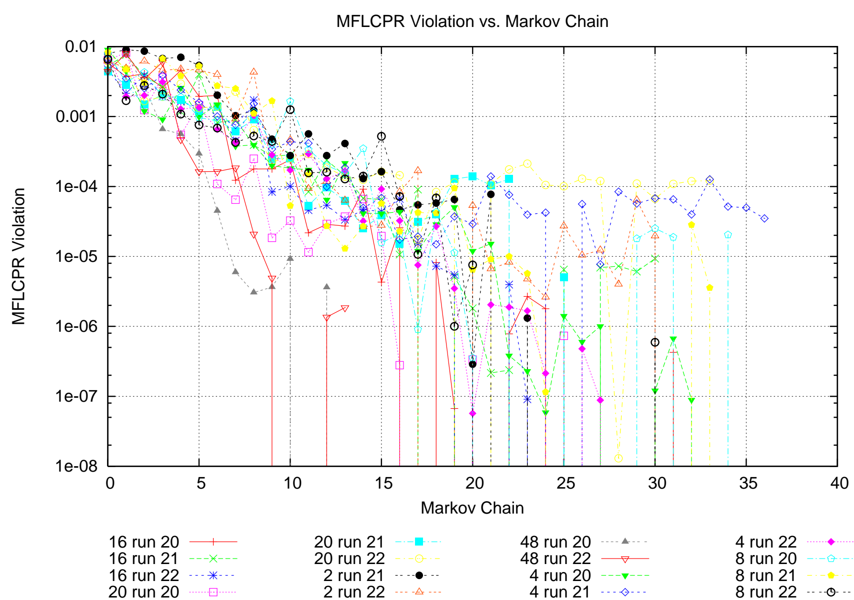


Figure 3.11: Depth versus Width Markov Chain Averaged MFLCPR

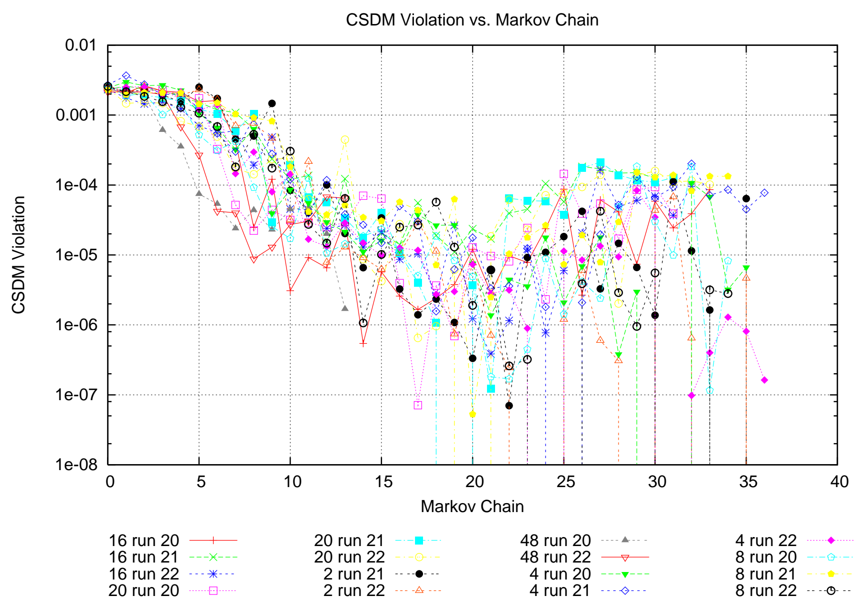


Figure 3.12: Depth versus Width Markov Chain Averaged CSDM

المنارة للاستشارات
www.manaraa.com

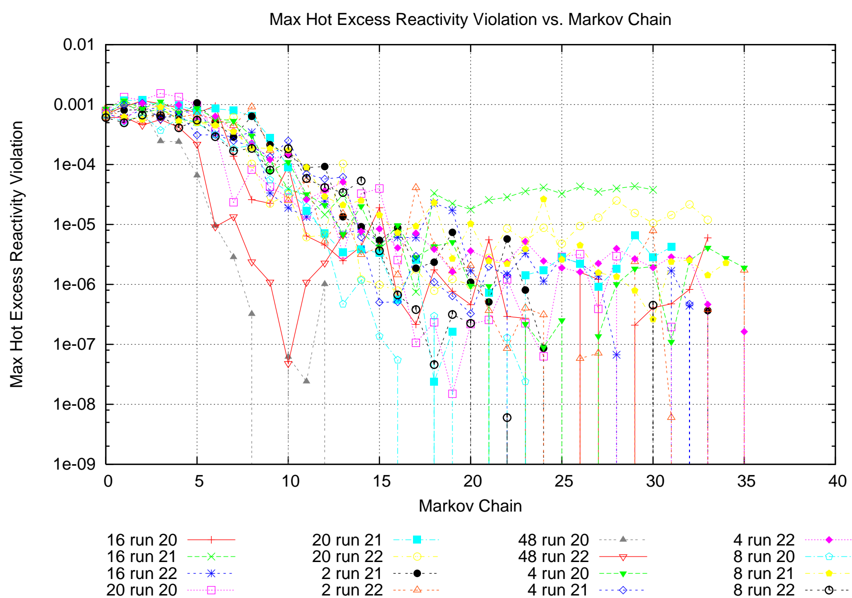


Figure 3.13: Depth versus Width Markov Chain Averaged Max HX

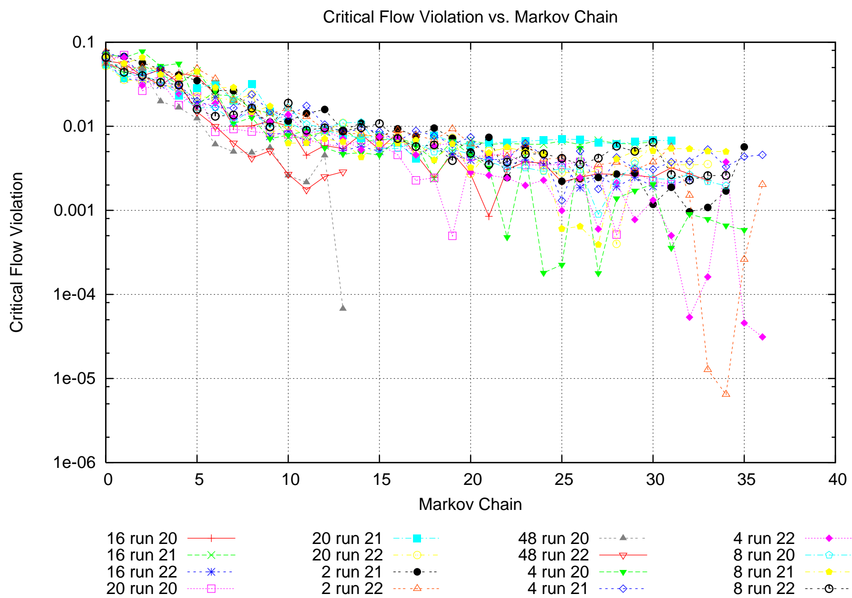


Figure 3.14: Depth versus Width Markov Chain Averaged Flow Violation

www.manaraa.com

## 3.6 Stochastic versus Deterministic Branching

The results presented to this point have utilized a deterministic, best-solution method to determine which solution to carry forward from the annealing temperature update step. One concern with this method is that it may cause the solution to become trapped in an unfavorable local minimum solution. As described in chapter 2, the binary branching mode selects the starting solution for the next Markov chain by pairing up the best solution found on each processor and iteratively selecting one member of each pair via the Metropolis criterion until a single solution remains. This algorithm was tested using a set of trials similar to those above. The settings used are presented below in table 3.10, with results in figures 3.15 through 3.21 and table 3.11.

Table 3.10: Binary Branching Settings

| # proc. | *llngth* | *lchain* | *ltran* | *lsurv* | CRP Update Threshold |
|---|---|---|---|---|---|
| 2 | 48,000 | 600 | 300 | 2500 | 1,000 |
| 4 | 48,000 | 300 | 150 | 1250 | 1,000 |
| 8 | 48,000 | 150 | 75 | 625 | 1,000 |
| 16 | 48,000 | 75 | 37 | 300 | 1,000 |
| 20 | 48,000 | 60 | 30 | 250 | 1,000 |
| 48 | 48,000 | 60 | 30 | 150 | 1,000 |

Table 3.11: Binary Branching Speedup Results

| # of Processes | # of Runs | Avg. # of Samples | Average Runtime[hr] | End-of-Cycle % Flow Change | Speedup | Efficiency |
|---|---|---|---|---|---|---|
| 2 | 2 | 48,690 | 95.7 | -10.7± 0.9% | 1.4 | 71.4% |
| 4 | 2 | 48,099 | 43.0 | -10.6± 0.0% | 3.1 | 76.7% |
| 8 | 4 | 48,431 | 20.4 | -10.4± 0.4% | 6.6 | 81.9% |
| 16 | 2 | 48,188 | 9.5 | -10.1± 0.0% | 14.3 | 89.6% |
| 20 | 2 | 48,727 | 9.4 | -10.1± 0.0% | 14.1 | 70.5% |
| 48 | 3 | 50,541 | 4.2 | -10.0± 0.0% | 32.6 | 67.9% |

The results from this series of tests indicate that for these settings, the optimization performance of the binary branching algorithm is not significantly different than for the synchronous algorithm. There is, however, a speedup and performance penalty compared to the synchronous best-solution algorithm (particularly for few processor cases). These conclusions may not apply for different settings or reactor models.

المنارة للاستشارات
www.manaraa.com

Table 3.12: Binary Branching Optimization Results

| # of Processes | Constraint Violation (x1,000,000) MFLPD | MAPRAT | MFLCPR |
|---|---|---|---|
| 2 | 0.00± 0.00 | 0.00 ± 0.00 | 0.00± 0.00 |
| 4 | 166.78± 39.49 | 3.13 ± 4.43 | 35.46± 50.15 |
| 8 | 90.18± 180.36 | 0.00 ± 0.00 | 17.61± 35.21 |
| 16 | 270.95± 86.34 | 59.28 ± 83.84 | 727.71± 1,029.13 |
| 20 | 86.67± 122.56 | 13.86 ± 19.61 | 72.60± 102.67 |
| 48 | 62.82± 48.84 | 0.00 ± 0.00 | 271.80± 470.77 |

| CRP Threshold | Constraint Violation (x1,000,000) CSDM | MAXHX | Critical Flow |
|---|---|---|---|
| 2 | 0.00± 0.00 | 88.54 ±125.22 | 6,150.51± 3,077.29 |
| 4 | 398.37± 35.58 | 0.00 ± 0.00 | 8,419.31± 7,016.13 |
| 8 | 70.13± 91.35 | 20.20 ± 40.40 | 7,965.06±11,011.85 |
| 16 | 518.92± 733.86 | 0.00 ± 0.00 | 5,871.80± 976.91 |
| 20 | 0.00± 0.00 | 0.00 ± 0.00 | 8,903.24± 5,299.84 |
| 48 | 859.91±1,489.41 | 0.00 ± 0.00 | 4,802.87± 4,522.97 |

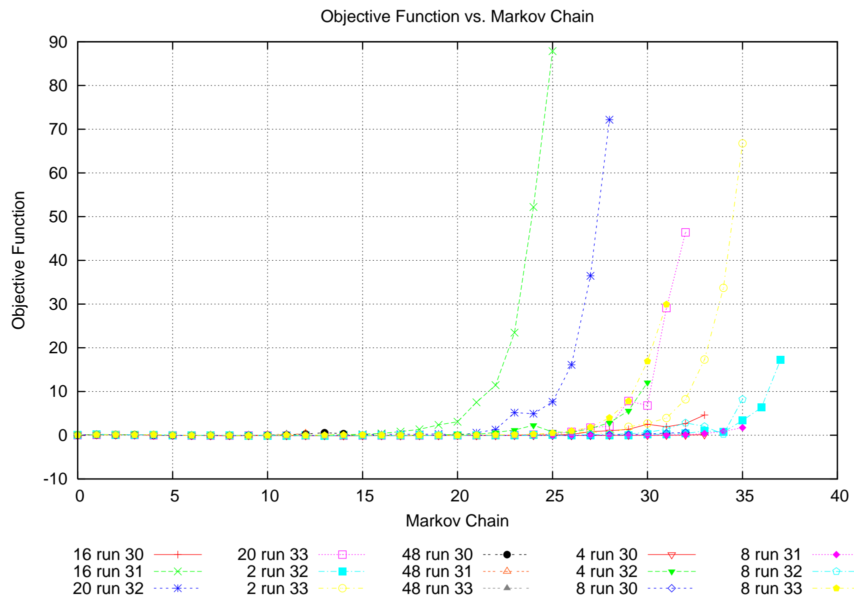


Figure 3.15: Binary Branching Markov Chain Averaged Objective Function

المنارة للاستشارات
www.manaraa.com

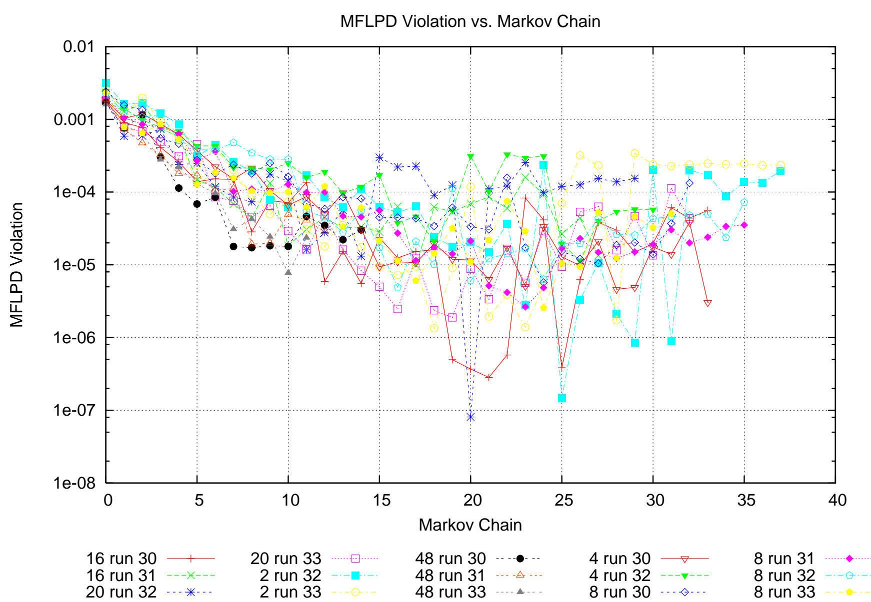


Figure 3.16: Binary Branching Markov Chain Averaged MFLPD

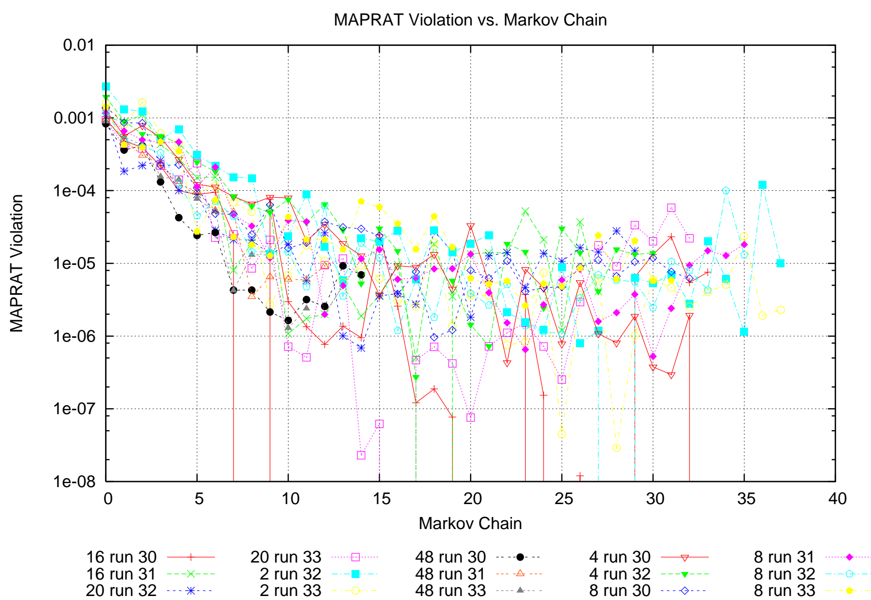


Figure 3.17: Binary Branching Markov Chain Averaged MAPRAT

www.manaraa.com

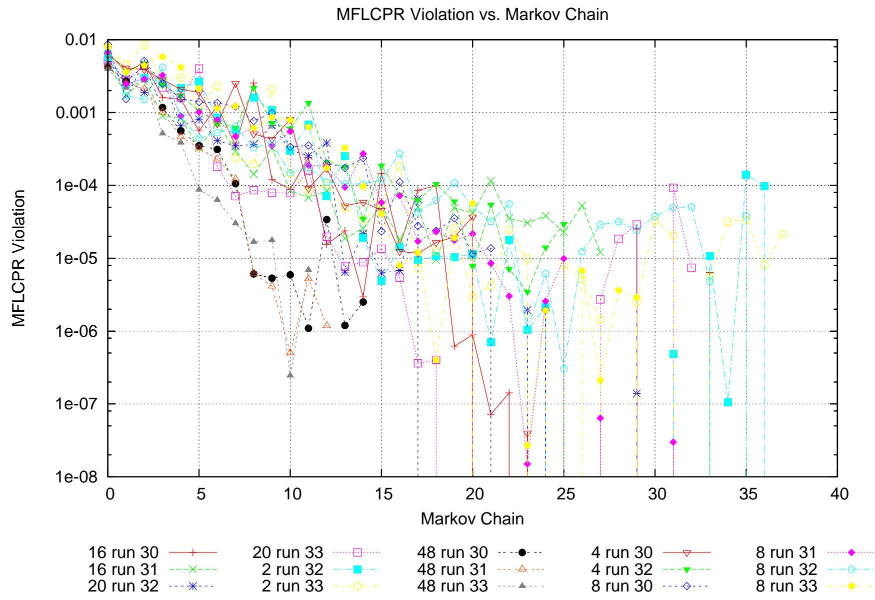


Figure 3.18: Binary Branching Markov Chain Averaged MFLCPR

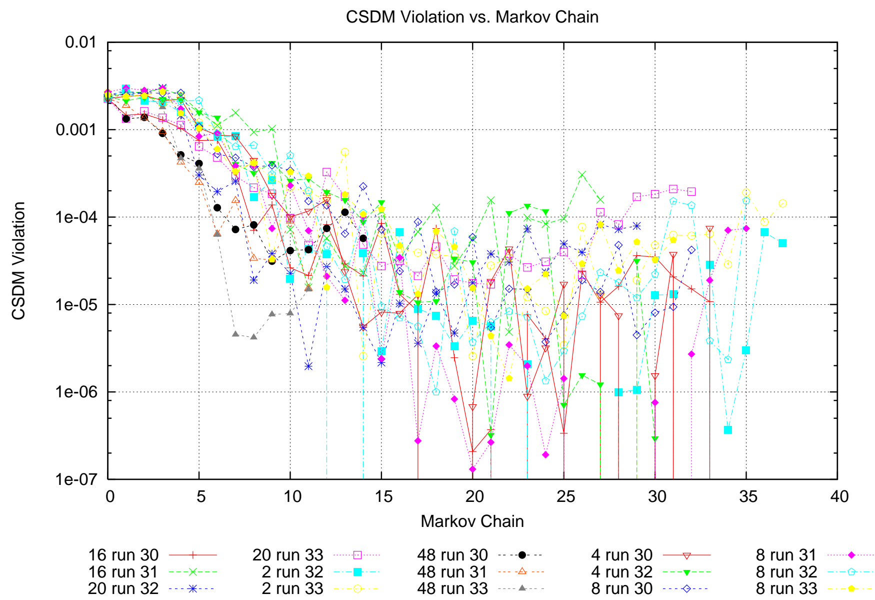


Figure 3.19: Binary Branching Markov Chain Averaged CSDM

www.manaraa.com

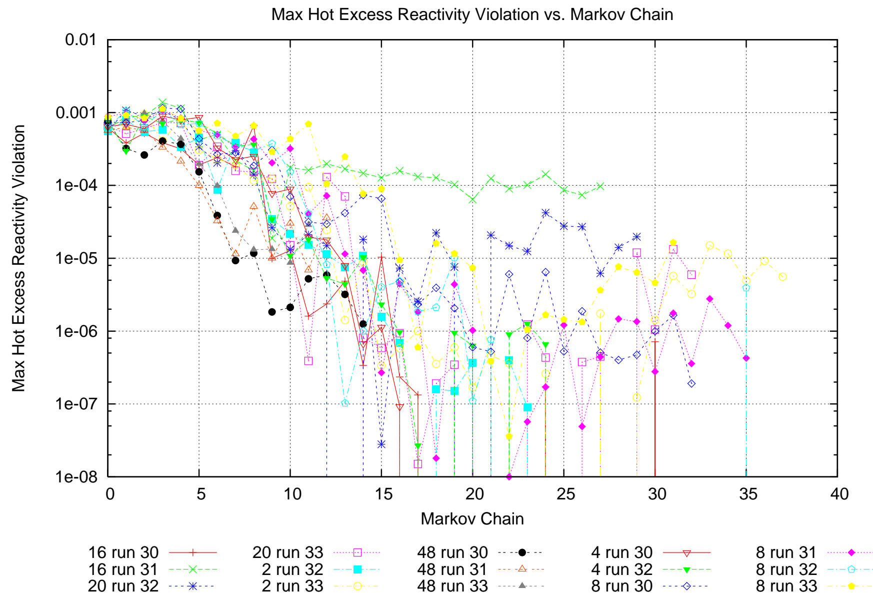


Figure 3.20: Binary Branching Markov Chain Average Max HX

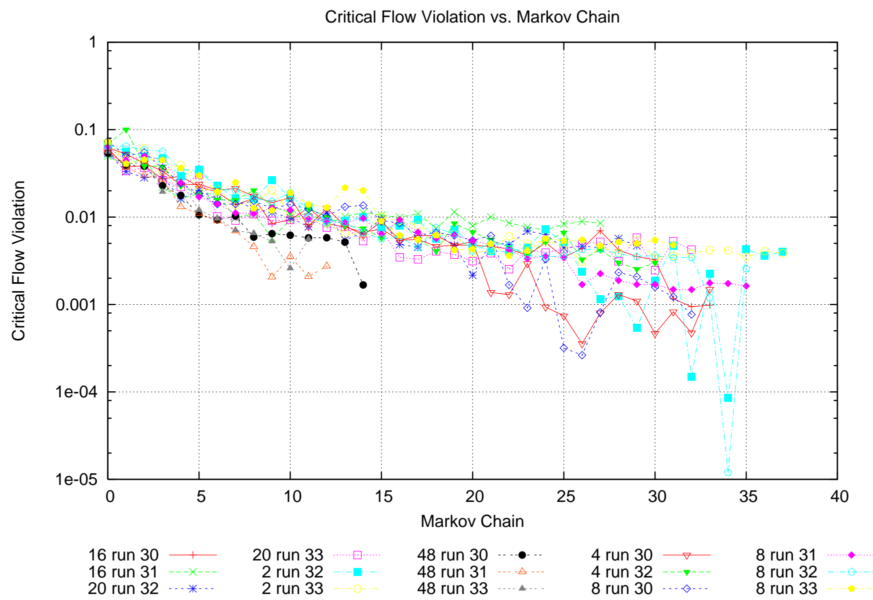


Figure 3.21: Binary Branching Markov Chain Average Flow Violation

www.manaraa.com

## 3.7 Parallel Performance

Several factors are important when considering the performance of a program. The achievable parallel speedup and efficiency are of primary concern here, but other factors such as memory usage and communications overhead can also have significant beneficial or detrimental effects. For example, many parallel clusters are composed of loosely coupled workstations or desktop computers. These systems are quite limited both in the amount of memory they are likely to possess and in the available communication bandwidth between machines.

To the engineer designing a reload core, the most important performance factors are likely to include robustness, ease-of-use and execution time. The first two of these factors are not considered here, as this is currently a developmental code. Execution time is listed, along with speedup and efficiency, next to the relevant test cases.

### 3.7.1 Communications Overhead

In many parallel numerical algorithms the amount of time spent on interprocess communication imposes limitations on the scalability of the method. Between the computational intensity of the core simulator, the infrequent need for interprocess communication and the high bandwidth of the BladeCenter network communications, overhead does not play a significant limiting role in the performance or scalability of this program.

### 3.7.2 Memory Usage

The overall memory usage for the test cases was small enough that it should pose no great memory management concerns. Typical memory usage was less than 250MB per process, which is well below the 2GB per node limit of available memory on the Henry2 cluster. Of greater concern is the disk space required for the various output files. Certain output edit options, such as the writing out of every loading pattern generated, can easily generate single files nearing one gigabyte in size. Two strategies are taken to address this issue. First, the selection of output edits is carefully chosen to avoid those that would exceed the available storage. Second, all parallel input and output is stored on the local scratch space of the parallel computing nodes. This has the dual benefit of minimizing network

المنارة للاستشارات
www.manaraa.com

communications during a run and avoiding any potential file access conflicts that might arise from multiple processes attempting simultaneous access. These files are automatically relocated to a central directory when the program exits.

www.manaraa.com

# Chapter 4

# Conclusions and Recommendations

The task designing a reload core for a commercial power reactor is a complicated and involved process, involving many interrelated tasks that start several years before the fuel enters the reactor. Operational and design decisions made during previous cycles can have long-lasting consequences to the fuel condition and usability. Additional complicating factors such as leaking fuel cladding or unplanned outages can further disrupt previous planning. For these and other reasons, automated analysis tools such as the FORMOSA-B code are of great potential utility to engineers seeking to find the best reload design in the most efficient manner. The development of a parallel optimization capability in the FORMOSA-B code further enhances its usefulness to engineers by reducing computational turnaround times while expanding the search coverage. The parallel communication methods used in this program are based on the MPI standard for interprocess communication, thus ensuring that the code can be readily adapted to the wide variety of computational clusters in common use in industry.

In tests carried out on an IBM BladeCenter cluster using up to 48 processors, the parallel algorithm exhibited a speedup factors exceeding 32 compared to serial computations on the same system. Parallel efficiencies range from 95% down to 68% as the number of processors increases. While varying the CRP Update Threshold appears to have very little effect on solution convergence and program run-times, the use of periodically optimized control rod programs is necessary to achieve good loading pattern optimization results.

Finally, only a coarse cross-cut of optimization parameters and reactor models were

المنارة للاستشارات
www.manaraa.com

tested in this study. It is likely that further tuning of the cooling schedule and operational parameters will yield even higher performance.

## 4.1 Future Work

As with nearly all projects of this size, there remains a great deal of analysis and development to be performed. These fall into three general categories: testing of the algorithm as it is, development of the algorithm, and general program updates.

Parallel `FORMOSA-B` testing so far has utilized only a small, 1911MWth BWR/4 model. In order to fully exercise the capabilities of the parallel algorithm, it should be tested with a larger, more complicated model, such as a full-core, 800-assembly BWR/6 model. Furthermore, testing so far has focused on a rather narrow set of optimization objectives. It would be worthwhile to test the parallel performance with objectives other than end-of-cycle flow minimization.

One peculiarity associated with stochastic optimization algorithms is their ability to return different results for different runs. Indeed the variability of results between runs is often larger than the result values themselves. Many of the tests presented here utilize only a handful of trials with each parameter set. By examining a larger sample size, more rigorous conclusions may be reached.

One difficulty often encountered when dealing with massively parallel codes is that faults and errors that were previously seldom, nuisance occurrences can happen with regularity. This is the case for both software glitches and hardware failures. For this reason it would be useful to upgrade the error detection and handling methods within the code to prevent and contain failures before any data are lost. Furthermore, a checkpointing capability would be quite useful. In this way, if a hardware or network failure interrupts execution of a job, it may be restarted later with minimal loss of data.

Finally, there are several other variations of the parallel simulated annealing algorithm that appear promising yet have not been implemented and tested. Among these are the asynchronous parallel simulated annealing, which is largely implemented but as yet untested, and the Mixing-of-States algorithm, which seems to offer improved scalability to large numbers of processors.

المنارة للاستشارات
www.manaraa.com

# BIBLIOGRAPHY


[1] Emile Aarts and Jan Korst. *Simulated Annealing and Boltzmann Machines.* Wiley - Interscience Series in Discrete Mathematics and Optimization. John Wiley and Sons, 1989.

[2] Robert Azencott, editor. *Simulated Annealing Parallelization Techniques.* John Wiley & Sons, Inc., 1992.

[3] Jonathan N. Carter. Genetic algorithms for incore fuel management and other recent developments in optimization. *Advances In Nuclear Science and Technology*, 25:113–154, 1997.

[4] Electric Power Research Center. *FORMOSA-B Code Methodology and Usage Manual - Version 3.2.* North Carolina State University, 2004.

[5] King-Wai Chu, Yuefan Deng, and John Reinitz. Parallel simulated annealing by mixing of states. *Journal of Computational Physics*, (148):646–662, 1999.

[6] Intel Corporation. *Intel Fortran Programmer's Reference.* Intel, 2003.

[7] Platform Computing Corporation. *Running Jobs with Platform LSF*, June 2002.

[8] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 1.3.* University of Tennessee, Knoxville, 2008.

[9] M.D. Huang, F. Romeo, and A. Sangiovanni-Vincentelli. An efficient general cooling schedule for simulated annealing. In *IEEE 1986 International Conference On Computer-Aided Design*, pages 381–384. IEEE, 1986.


المنارة للاستشارات
www.manaraa.com

[10] Intel Corporation. *Intel Fortran Compiler for Linux Systems User's Guide*, 2003. Document No. FL-710-01.

[11] Atul A. Karve and Paul J. Turinsky. Formosa-b: A boiling water reactor in-core fuel management optimization package ii. *Nuclear Technology*, 131(1):48–68, July 1999.

[12] Atul A. Karve and Paul J. Turinsky. Formosa-b: A boiling water reactor in-core fuel management optimization package iii. *Nuclear Technology*, 135(3):241–251, September 2001.

[13] Doddy Febrian Kastanya. *Implementation of a Newton-Krylov Iterative Method to Address Strong Non-Linear Feedback Effects in FORMOSA-B BWR Core Simulator*. PhD thesis, North Carolina State University, 2002.

[14] Paul M. Keller. Adaptively constrained multiobjective genetic algorithms for incore fuel management optimization. In *Transactions of the American Nuclear Society*, volume 92, pages 610–611, June 2005.

[15] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[16] David J. Kropaczek. *In-Core Nuclear Fuel Management Optimization Utilizing Simulated Annealing*. PhD thesis, North Carolina State University, 1996.

[17] David J. Kropaczek. Concept for multi-cycle nulcear fuel optimization basd on parallel simulated annealing with mixing of states. In *International Conference on the Physics of Reactors "Nuclear Power: A Sustainable Resource"*, 2008.

[18] Hyun Chul Lee, Hyung Jin Shim, and Chang Hyo Kim. Parallel computing adaptive simulated annealing scheme for fuel assembly loading pattern optimization in pwrs. *Nuclear Technology*, 135(2):39–50, July 2001.

[19] Soo-Young Lee and Kyung Geun Lee. Synchronous and asynchronous parallel simulated annealing with multiple markov chains. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):993–1008, October 1996.

المنارة للاستشارات

www.manaraa.com

[20] Samir W. Mahfoud and David E. Goldberg. Parallel recombinative simulated annealing: A genetic algorithm. *Parallel Computing*, 21:1–28, 1995.

[21] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953.

[22] Brian R. Moore. *Higher Order Generalized Perturbation Theory for BWR In-Core Nuclear Fuel Management Optimization*. PhD thesis, North Carolina State University, 1996.

[23] Brian R. Moore, Paul J. Turinsky, and Atul A. Karve. Formosa-b: A boiling water reactor in-core fuel management optimization package. *Nuclear Technology*, 126(1):153–169, March 1998.

[24] Office of Nuclear Reactor Regulation. Standard technical specifications general electric plants, bwr/4. Technical Report NUREG-1433, U.S. Nuclear Regulatory Commission, April 1995.

[25] Office of Nuclear Reactor Regulation. Standard technical specifications general electric plants, bwr/6. Technical Report NUREG-1434, U.S. Nuclear Regulatory Commission, April 1995.

[26] Richard Rhodes. *The Making of the Atomic Bomb*. Sloan Science Series. Simon & Schuster Adult Publishing Company, 1995.

[27] Eric Sills. Getting started with the ibm bladecenter linux cluster (henry2) at nc state. http://www.ncsu.edu/itd/hpc/Documents/BladeCenter/GettingStartedbc.php, December 2008.

[28] S.R. Specker, L.E. Fennern, R.E. Brown, R.L. Crowther, and K.L. Stark. Bwr/6 general description of a boiling water reactor. Technical report, General Electric Company, 1980. Section 3, Appendix: Control Cell Core Improved Design.

[29] William Stallings. *Computer Organization and Architecture*. Pearson Prentice Hall, Upper Saddle River, NJ, seventh edition, 2006.

المنارة للاستشارات
www.manaraa.com

[30] P.J.M. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications.* Mathematics and Its Applications. D. Reidel Publishing Company, Dordrecht, Holland, 1987.

[31] David Watts, Randall Davis, Illa Kroutov, and Kevin Galloway. *IBM BladeCenter Products and Technology.* IBM International Technical Support Organization, 2008.

المنارة للاستشارات

www.manaraa.com