Annals of Nuclear Energy 91 (2016) 135–141

Contents lists available at ScienceDirect

Annals of Nuclear Energy

journal homepage: www.elsevier.com/locate/anucene

Parallel GPU implementation of PWR reactor burnup

A. Heimlich [1], F.C. Silva *, A.S. Martinez *

Universidade Federal do Rio de Janeiro, Programa de Engenharia Nuclear Ilha do Fundão, 21945-970, P.O. Box 68509, Rio de Janeiro, RJ, Brazil

ARTICLE INFO

Article history:
Received 14 October 2015
Received in revised form 4 January 2016
Accepted 8 January 2016
Available online 29 January 2016

Keywords:
GPU
Parallel processing
Nuclear fuel burnup
Depletion
Multicore
Exponential matrix

ABSTRACT

This paper surveys three methods, implemented for multi-core CPU and graphic processor unit (GPU), to evaluate the fuel burn-up in a pressurized light water nuclear reactor (PWR) using the solutions of a large system of coupled ordinary differential equations. The reactor physics simulation of a PWR reactor spends a long execution time with burnup calculations, so performance improvement using GPU can imply in better core design and thus extended fuel life cycle. The results of this study exhibit speed improvement exceeding 200 times over the sequential solver, within 1% accuracy.

1. Introduction

A pressurized light water nuclear reactor (PWR) refueling typically replaces about a third of the spent fuel every twelve to eighteen months, depending on fuel burnup. Burnup indicates whether a fuel element must be replaced or reallocated. Refueling is a combinatorial optimization problem, and as such, the number of evaluated candidate solutions is a function of time and of computational resources.

Computational tools to model the reactor core are used to evaluate its reactivity, spatial neutronic behavior, power distribution, isotopic inventory and fuel burnup. The fuel burnup can be represented by a system of first order, ordinary, coupled differential equations (ODE), accounting for all fissionable actinides and fission fragment yields for the radioactive reaction chains under analysis. This approach creates a large system of equations that can be represented in matrix form Hairer and Wanner (1996).

Recent advancement in graphics processor units (GPU) gave a boost in parallel computing performance. Use of GPUs can improve the performance of nuclear reactor physics calculations by providing T Flops (10^{12} floating point operations) of computational processing power with low cost per watt using desktop computers. The use of graphic processors and hybrid programming for nuclear physics problem solving shows great performance improvement (Heimlich et al., 2011; Waintraub et al., 2011; Pereira et al., 2013) over sequential procedures.

The main objective of the present study is to provide faster, expansible and reliable methods to calculate fuel burnup in PWR reactors. The original contribution of this work is the development of three methods to obtain the solution of a large system of ordinary coupled differential equations for execution in graphic processor units (GPU) and in multicore CPU using parallel programming techniques. The performances of the three methods are compared to each other and their respective solutions to a reference one obtained by a sequential solver.

The first method is Runge–Kutta–Fhelberg's. It employs a classic approach to solve a differential equation system. The second is Jacobi Collocation method, where the matrix exponential is approximated by Gauss quadrature using Jacobi polynomials. The final method employs matrix exponential rational approximation using Padé's diagonal method.

2. Theory

The nuclide concentration in nuclear reactor's fuel is described by a system of differential equations which relate the production and consumption of each nuclide for each radioactive chain reaction. These changes in nuclide concentration are produced by decay, fission, radioactive capture and scattering and are described by Bateman's equation (Bateman, 1910). Eq. (1) represents the abundance variation for nuclide k.

* Corresponding authors.
E-mail addresses: adino.heimlich@ien.gov.br (A. Heimlich), fernando@nuclear.ufrj.br (F.C. Silva), aquilino@lmp.ufrj.br (A.S. Martinez).
[1] Principal corresponding author.

http://dx.doi.org/10.1016/j.anucene.2016.01.010
0306-4549/

$$\frac{dN_k^x}{dt}(t) = \overbrace{\sum_{i=1,i\neq k}^{\text{Actinide-series}} N_i^x(t)\sum_{g=1}^{G}\left(\sigma_{\gamma,i}^{g,x}(t) - \sigma_{f,i}^{g,x}(t) + \sigma_{(n,2n),i}^{g,x}(t) + \right)\phi_x^g(t)}$$
$$- \overbrace{\sum_{y=1} \lambda_{i,y} N_i^x(t)}^{\text{Decay Fractions}} + \overbrace{\sum_{z\neq i} \lambda_z N_z^x(t)}^{\text{Production Fractions}} \quad (1)$$

The abundance variation of nuclide l induced by fission of actinides is described by Eq. (2).

$$\frac{dN_l^x}{dt}(t) = \overbrace{\sum_{i=1,i\neq l}}^{\text{Fission Yields}} \left(N_i^x(t)\sum_{g=1}^{G}\left(\Gamma_{i,l}^g \sigma_{f,i}^{g,x}(t) - \sigma_{\gamma,i}^{g,x}(t)\right)\phi_x^g(t)\right)$$
$$- \lambda_l N_l^x(t) \quad (2)$$

Variables x, t and g represent the spatial position, time and energy group respectively. $N_k^x(t)$ represents actinide k concentration. Cross sections are represented by $\sigma_{f,i}^{g,x}(t), \sigma_{\gamma,i}^{g,x}(t)$ and $\sigma_{(n,2n),i}^{g,x}(t)$. Decay constants are $\lambda_{i,y}$ in reaction branch y. The neutron flux is $\phi_x^g(t)$ and finally $\Gamma_{i,l}^g$ represents nuclide l yield from fission reaction of actinide i.

Differential equations systems (1) and (2) can be represented in matrix form by Eq. (3) with solution in Eq. (4). This is a well-known problem (Bellman et al., 1970) in differential equation theory and its solution can be obtained by matrix exponential methods in at least nineteen ways (Moler and Van Loan, 2003).

$$\frac{d\vec{N}(t)}{dt} = \mathbf{B}\cdot\vec{N}(t), \quad \vec{N}(0) = \vec{N}_0 \quad (3)$$

Vector $\vec{N}(0)$ represents the initial nuclide concentration and $\mathbf{B}$ is the depletion matrix.

$$\vec{N}(t) = e^{\mathbf{B}t}\cdot\vec{N}(0) \quad (4)$$

Thus, the evolution of nuclide concentrations can be evaluated using the recursive procedure below and variable $\Delta t = t_n - t_{n-1}$ is the burnup step.

$$\vec{N}(t_n) = e^{\mathbf{B}\Delta t}\cdot\vec{N}(t_{n-1}) \quad (5)$$

This study employed the simulation of reactivity and spatial power distribution in the reactor core determinated by CNFR (Alvim et al., 2010; da Silva et al., 2010) (acronym for *National Physics Reactor Code* in Portuguese). CNFR simulates the behavior of PWR reactors by solving a cartesian, tridimensional, double energy group, steady state approximation of the neutron diffusion equation. The solution is based on nodal expansion method (Finnemann et al., 1977) where the reactor is spatially divided in as many as four thousand *nodes*. The simulator also evaluates the fuel burnup and overall calculations with thermohydraulics feedback.

The CNFR code was implemented in FORTRAN for use in a single core processor, thus the burnup algorithm must be executed sequentially node by node. The sequential procedure to evaluate the burnup computes at least 3000 nodes, each one of the nodes represented by a system of 37 differential equations which in turn implies in a matrix operator with the same number of lines.

CNFR evaluates reactor core burnup and inventory by solving only the most important radioactive reaction chains needed to maintain the stable production of energy in the core.

GPU hardware architecture achieves high performances when used to process very large matrices and vectors (Bell and Garland, 2008). The linear algebra operator *direct sum* $\oplus$ (Gantmacher, 1959) was used to create this large sparse matrix operator. The depletion and inventory operator built to calculate the burnup and inventory in a GPU was created by joining each individual node operator, represented by a matrix-valued operator, in a large sparse matrix operator, with over one hundred thousand lines and at least five millions non zero elements. Fig. 1 shows the matrix associated to the depletion operator for one node and eight nodes matrices.

The next sections explain the building of this large matrical operator and the implemented methods.

2.1. Depletion matrix operator

The GPU based constructor, with linear algebra manipulation, storage and input/output procedures, was developed using *Compute Unified Device Architecture* (CUDA) (Luebke, 2008) and C++ *Standard Template Library* (STL) to provides access and execute sparse linear algebra routines using library cuSparse (Naumov et al., 2010) and cuBlas (Nvidia, 2008).

Eq. (3) shows the ODE system written in matrix form. The matrix operator $\mathbf{B}$ must be rebuild in each step of the burnup evaluation because the sparsity of the matrix operator $\mathbf{B}$ changes with spatial position and actinide concentration. This procedure builds the operator $\mathbf{B}$ with CNFR data arrays, including microscopic cross sections (total, absorption, capture) of all nuclides, actinides microscopic cross sections (fission, $(n,2n)$), fission yields and neutron flux.

The matrix operator is built using the *Compressed Oriented Ordinates* (COO) format and then converted to *Compressed Sparse Row* (CSR) format to improve performance (Bell and Garland, 2008). The building of the operator can be extended to evaluate decay chains and fission yields for other actinide series, *e.g.*, the Th^{232} chain.

The matrix container defined above is assigned by each method operator building in this work. These operators employ the sparse matrix construct to obtain a function, whether as *Sparse Matrix Vector Multiplications* (SpMv) in Runge–Kutta–Fhelberg or *Power of Matrix* in Jacobi Collocation and Padé's Approximation methods.

2.2. Runge–Kutta methods

Runge–Kutta methods (Hairer et al., 1989) are reliable and widely used to find a solution to an initial value problem (Dormand and Prince, 1980) of the form:

$$y'(x) = f(x, y(x)), \quad y(x_0) = y_0 \quad (6)$$

and to obtain an approximation of $y(x)$ using a truncated Taylor series. This method finds an approximation with

$$y_n = y_{n-1} + h\cdot\sum_{i=1}^{s} b_i k_i + O(h^{s+1}). \quad (7)$$

Similarly, the ODE system can be solved on vector form by the Eq. (8).

$$\vec{y}_n = \vec{y}_{n-1} + h\cdot\sum_{i=1}^{s} b_i \vec{k}_i + O(h^{s+1}) \quad (8)$$

S is the order of the method and $\vec{k}_i$ is given by Eq. (9)

$$\vec{k}_i = \mathbf{f}\left(\vec{y}_{n-1} + h\cdot\sum_{j=1}^{s} a_{ij}\vec{k}_j, t_n + c_i h\right) \quad (9)$$

The coefficients a_{ij} are defined by a quadrature rule and $\mathbf{f}$ is the matrix operator.

2.2.1. Runge–Kutta–Fhelberg

In this approach, Shampine (1977) uses an adaptive time-step ($h = h\cdot\tau$) correction, calculated by the fifth-order and fourth-order approximations difference. The Eq. (10) shows the vectors $\vec{k}_i$ calculi and the number of SpMv operations.