Annals of Nuclear Energy 99 (2017) 174–182

Contents lists available at ScienceDirect

Annals of Nuclear Energy

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

HPC implementation in the time-dependent neutron diffusion code AZKIND

Andrés Rodríguez Hernández [a], Armando Miguel Gómez Torres [a,*], Edmundo del Valle Gallegos [b,1]

[a] Instituto Nacional de Investigaciones Nucleares, Carr. Mexico-Toluca s/n, La Marquesa, C.P. 52750 Ocoyoacac, Edo. de México, Mexico
[b] Escuela Superior de Física y Matemáticas, Instituto Politécnico Nacional, San Pedro Zacatenco, C.P. 07738 Cd. de México, Mexico

ARTICLE INFO

Article history:
Received 30 June 2016
Accepted 25 August 2016
Available online 7 September 2016

Keywords:
HPC, high performance computing
NFEM, nodal finite element method
Parallel computing
GPU, Graphics Processing Unit
Neutron diffusion theory

ABSTRACT

This article presents a summary of the development of the computer code AZKIND. This code is based on multi-group neutron diffusion theory, where the kinetics equations include delayed neutron precursors. For space discretization a Galerkin process is applied using a nodal finite elements method, and the well-known θ-method is used for time discretization. A high performance computing methodology was implemented in AZKIND to solve the resulting linear algebraic system $A\vec{v} = \vec{b}$ where numerical solution of large algebraic systems representing full nuclear reactor cores is achieved with an acceleration technique based on the open source linear algebra PARALUTION library. This implementation allows AZKIND threading into a GPU thousands of arithmetic operations for parallel processing. The acceleration is demonstrated with the use of different nuclear fuel arrays resulting in extremely large matrices, getting a speedup ratio up to 48. Finally, the acceleration capabilities of the HPC implementation are presented in the simulation of a power transient in an actual boiling water reactor core.

1. Introduction

Parallel computing is the computer science discipline that deals with the system architecture and software issues related to the concurrent execution of applications. The interest in parallel computing dates back to the late 1950s, with advancements surfacing in the form of supercomputers throughout the 60s and 70s. Starting in the late 80s, clusters competed and eventually displaced multiple parallel processors (MPP) for many applications. Today, clusters are the workhorse of scientific computing and are the dominant architecture in data centers. Parallel computing has become a mainstream based on multi-core processors.

In the particular case of the nuclear industry, since the late 50's, the need to perform nuclear safety analyses was essential, mainly for commercial nuclear power plants. In this sense, the application of scientific computing calculations has made possible these safety analyses, but always struggling with computer capabilities. In the former efforts, the main goal was to get a static or dynamic solution to a set of partial differential equations for neutron diffusion and neutron transport with technology and methods present in those years. For this huge task, numerical techniques were used with finite differences, finite elements, and nowadays, nodal finite elements. No matter which numerical method is used, the analyst always faces the problem of solving an extremely large algebraic system that challenges computer capabilities. It is desirable for the nuclear safety analyst to obtain the results of each reactor simulation in a relatively short time.

In the last few years, the technological developments of supercomputers or high performance computer equipment have made possible the use of supercomputing in many scientific areas, including nuclear applications. A proper combination of parallel computing software, like already developed linear algebra libraries, and a specific project can result in a computational platform to simulate nuclear reactor states in relatively quick times.

This paper is organized as follows, Section 2 summarizes the use of a nodal finite element method applied to a set of coupled partial differential equations to generate algebraic systems to be solved for nuclear reactors. Section 3 describes the characteristics of the algebraic system and matrix structure. The implementation of high performance computing is presented in Section 4 with the use of the PARALUTION library in the neutron kinetics code AZKIND. A verification of the modern version of AZKIND for reactor steady state calculations with serial processing is contained in Section 5. Section 6 describes the construction of different sizes of matrices, from small up to extremely large matrices, with some verification tests for parallel versus serial computing. A demonstra-

* Corresponding author.
E-mail addresses: armagotoro@hotmail.com, armando.gomez@inin.gob.mx (A.M. Gómez Torres).
[1] On sabbatical leave at the Instituto Nacional de Investigaciones Nucleares.

http://dx.doi.org/10.1016/j.anucene.2016.08.019
0306-4549/

tion of the acceleration of computer calculations is shown in Section 7 with the use of different hardware capabilities, followed by the simulation of a reactor power transient. Finally, conclusions and an outlook of coming efforts are presented in Section 8.

2. Neutron diffusion theory and nodal methods

2.1. Multigroup time-dependent neutron diffusion equations

The development presented in this section is based on multigroup neutron diffusion theory. For G neutron energy groups and I_p delayed neutron precursor concentrations, the neutron diffusion kinetics equations are given by Eqs. (1) and (2) (Duderstadt and Hamilton, 1976):

$$\frac{1}{v^g}\frac{\partial}{\partial t}\phi^g(\vec{r},t) = \nabla\cdot D^g\nabla\phi^g(\vec{r},t) - \Sigma_R^g(\vec{r},t)\phi^g(\vec{r},t) + \sum_{\substack{g'=1\\ g'\neq g}}^{G}\Sigma^{g'\to g}\phi^{g'}(\vec{r},t) + (1-\beta)\chi^g\sum_{g'=1}^{G}\nu^{g'}(\vec{r},t)\Sigma_f^{g'}\phi^{g'}(\vec{r},t) + \sum_{i=1}^{I_p}\chi_i^g\lambda_i C_i(\vec{r},t) \quad (1)$$

$$g = 1,\ldots,G; \quad \forall(\vec{r},t)\in\Omega\times(0,T]$$

$$\frac{\partial}{\partial t}C_i(\vec{r},t) = \beta_i\sum_{g'=1}^{G}\nu^{g'}(\vec{r},t)\Sigma_f^{g'}(\vec{r},t)\phi^{g'}(\vec{r},t) - \lambda_i C_i(\vec{r},t) \quad (2)$$

$$i = 1,\ldots,I_p; \quad \forall(\vec{r},t)\in\Omega\times(0,T]$$

In addition to boundary conditions for neutron fluxes, initial conditions must be satisfied by neutron fluxes and neutron precursors functions. The parameters involved in the above equations are described in Duderstadt and Hamilton (1976).

2.2. Spatial discretization

The spatial discretization of these equations is strongly connected with the discretization of a nuclear reactor core of volume Ω. Representing the neutron flux and the precursor concentrations in terms of base functions defined over Ω, it is possible to write:

$$\phi^g(\vec{r},t) = \sum_{k=1}^{N_f}u_k(\vec{r})\phi_k^g(\vec{r},t); \quad g=1,\ldots,G; \quad \forall(\vec{r},t)\in\Omega\times(0,T]; \quad (3)$$

$$C_i(\vec{r},t) = \sum_{m=1}^{N_p}v_m(\vec{r})C_i^m(\vec{r},t); \quad i=1,\ldots,I_p; \quad \forall(\vec{r},t)\in\Omega\times(0,T]; \quad (4)$$

where N_f and N_p are the number of unknowns to be determined for neutron flux and delayed neutron precursors, respectively. Substituting expressions (3), (4) into (1), (2), and applying the Galerkin process for spatial discretization, as described in Rodríguez-Hernández (2002), the resulting algebraic system of equations can be expressed in matrix notation as follows,

$$\frac{1}{v^g}\boldsymbol{M}_f\frac{d}{dt}\boldsymbol{\phi}^g(t) = -\boldsymbol{K}^g\boldsymbol{\phi}^g(t) - \sum_{g'=1}^{G}\boldsymbol{S}^{g'\to g}\boldsymbol{\phi}^{g'}(t) + (1-\beta)\chi^g\sum_{g'=1}^{G}\boldsymbol{F}^{gg'}(t)\boldsymbol{\phi}^{g'}(t) + \sum_{i=1}^{I_p}\boldsymbol{F}^{gi}\boldsymbol{C}_i(t) \quad (5)$$

$$g = 1,\ldots,G; \quad \forall(\vec{r},t)\in\Omega\times(0,T]$$

$$\boldsymbol{M}_p\frac{d}{dt}\boldsymbol{C}_i(t) = \sum_{g'=1}^{G}\boldsymbol{P}^{ig'}(t)\boldsymbol{\phi}^{g'}(t) - \lambda_i\boldsymbol{M}_p\boldsymbol{C}_i(t); \quad (6)$$

$$i = 1,\ldots,I_p; \quad \forall(\vec{r},t)\in\Omega\times(0,T]$$

where, $\boldsymbol{\phi}^g(t) = \left[\phi_1^g(t),\ldots,\phi_{N_f}^g(t)\right]^T$ and $\boldsymbol{C}_i(t) = \left[C_i^1(t),\ldots,C_i^{N_p}(t)\right]^T$.

Section 2.3 explains the computation of the matrix elements with the use of nodal base functions for the method *RTN-0* to evaluate the integral expressions in Table 1 over each node Ω_h in the full domain Ω. See Fig. 1.

2.3. Nodal finite element method in spatial discretization

A particular nodal finite element (NFE) is characterized by the fact that for each cell (node) the function unknowns to be determined are the (00) Legendre moment (average) of the unknown function over each face of the node and the (000) Legendre moment over the cell volume.

Fig. 1a shows a physical domain Ω represented graphically after generating an *xyz* mesh. Fig. 1b shows a cuboid-type cell with directions through the faces: (x) Right, Left; (y) Near, Far; (z) Top, Bottom; and C for the average of the function over the cell volume.

Taking into consideration the general form to build up nodal schemes (Hennart, 1986), the moments of a function (at edges and cell) over a node like the one shown in Fig. 1b can be written for the nodal finite element method RTN-0 (***R**aviart-**T**homas-**N**édélec*).

In the NFE method RTN-0, the normalized zero order Legendre polynomials, defined over the unit cell $\Omega_{ijk} = [-1,1]\times[-1,1]\times[-1,1]$ and correlated to each physical cell $\Omega_h = \Omega_{ijk} = [x_i,x_{i+1}]\times[y_j,y_{j+1}]\times[z_k,z_{k+1}]$, are used to calculate the elements of the matrices in Eqs. (5) and (6). The matrix elements are quantified introducing the following nodal base functions (Hennart, 1986):

$$u_L^{00}(x,y,z) = -\frac{1}{2}(P_{100}-P_{200}); \quad u_R^{00}(x,y,z) = +\frac{1}{2}(P_{100}+P_{200});$$
$$u_N^{00}(x,y,z) = -\frac{1}{2}(P_{010}-P_{020}); \quad u_F^{00}(x,y,z) = +\frac{1}{2}(P_{010}+P_{020});$$
$$u_B^{00}(x,y,z) = -\frac{1}{2}(P_{001}-P_{002}); \quad u_T^{00}(x,y,z) = +\frac{1}{2}(P_{001}+P_{002});$$
$$u_C^{000}(x,y,z) = P_{000}-P_{200}-P_{020}-P_{002};$$

where $P_{lpq}(x,y,z) = P_l(x)P_p(y)P_q(z);\ l,p,q = 0,1,2$.

An extensive discussion on nodal diffusion methods can be found in reference (Grossman and Hennart, 2007) for space discretization using simplification approaches for calculating the moments over a node.

2.4. Discretization of the time variable

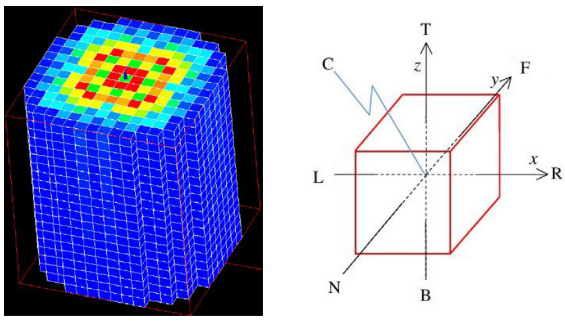
Once the spatial discretization is done, the θ-method (Rodríguez-Hernández, 2002) is applied for the discretization of the time variable appearing in the algebraic system given by (5) and (6). For the time integration over the interval $(0,T]$, this interval is divided in L time-steps $\Delta t_l = [t_l,t_{l+1}]$ and the following approach is assumed:

$$\int_{t_l}^{t_{l+1}} f(t)dt \cong h_l\left[\theta f_{l+1} + (1-\theta)f_l\right] \quad (7)$$

where $h_l = \Delta t_l, f_l = f(t_l), f_{l+1} = f(t_{l+1})$, and θ is the integration parameter.

Table 1
Matrix elements from the spatial discretization.

Matrix	Type	Dimension	Elements
M_f	Mass	$N_f \times N_f$	$m_{f,jk} = \int_\Omega u_j u_k d\vec{r}$
M_p	Mass	$N_p \times N_p$	$m_{p,lm} = \int_\Omega v_l v_m d\vec{r}$
K^g	Stiffness	$N_f \times N_f$	$k^g_{jk} = \int_\Omega D^g \nabla u_j \cdot \nabla u_k d\vec{r}$
$S^{g'\to g}$	Mass	$N_f \times N_f$	$s^{g'\to g}_{jk} = \int_\Omega \Sigma^{g'\to g} u_j u_k d\vec{r}$
$F^{gg'}$	Mass	$N_f \times N_f$	$f^{gg'}_{jk} = \chi^g \int_\Omega \nu^{g'} \Sigma^{g'}_f u_j u_k d\vec{r}$
F^{gi}	Mass	$N_f \times N_p$	$f^{gi}_{jm} = \lambda_i \chi^g_i \int_\Omega \nu_j u_m d\vec{r}$
$P^{ig'}$	Mass	$N_p \times N_f$	$p^{ig'}_{lk} = \beta_i \int_\Omega \nu^{g'} \Sigma^{g'}_f u_l \nu_k d\vec{r}$



(a) Domain Ω (b) Physical local node Ω_h

Fig. 1. Discretization of domain Ω.

In this way, when carrying out the time integration, both parameters θ_f and θ_p for neutron flux and delayed neutron precursors are considered. Depending on the values assigned, in the interval $[0,1]$, to the theta parameters θ_f and θ_p, different time integration schemes are generated (Rodríguez-Hernández, 2002), but they are not discussed here.

Applying the cited time integration method, the following set of equations are produced,

$$D_g^{j+1}\phi^g_{j+1} + \sum_{g'=1}^{G} T^{j+1}_{gg'}\phi^{g'}_{j+1} + \sum_{i=1}^{I_p} F_{gi} C_i^{j+1} = S^g_j; \qquad g = 1,\ldots,G; \tag{8}$$

$$\sum_{g'=1}^{G} P^{j+1}_{ig'}\phi^{g'}_{j+1} + \Lambda_i C_i^{j+1} = S^j_i; \quad i = 1,\ldots,I_p; \tag{9}$$

Eqs. (8) and (9) can be expressed as an algebraic system if the flux and precursors vectors, and the corresponding right-hand-side terms, are ordered as follows,

$$\Psi_{j+1} = [\phi^1_{j+1},\ldots,\phi^G_{j+1}, C_1^{j+1},\ldots,C_{I_p}^{j+1}]^T; \quad S_j = [S^1_j,\ldots,S^G_j, S^j_1,\ldots,S^j_{I_p}]^T$$

$$B_{j+1}\Psi_{j+1} = S_j; \qquad j = 0,1,2,\ldots;$$

The dimension of matrix B is $[N_f G + N_p I_p] \times [N_f G + N_p I_p]$ since the solution vector Ψ would give the neutron fluxes for the N_f domain's points for each group g, and the delayed neutron precursor concentrations for the N_p points for each i precursor group.

In order to reduce the algebraic system, a simplification is done by solving vector C_i^{j+1} from Eq. (9) and inserting it into Eq. (8), yielding the following expression:

$$D_g^{j+1}\phi^g_{j+1} + \sum_{g'=1}^{G} T^{f,j+1}_{gg'}\phi^{g'}_{j+1} = S^g_{f,j}; \tag{10}$$

The Eq. (10) can also be expressed as an algebraic system as follows, where the matrix dimension is now $[N_f G] \times [N_f G]$:

$$A_{j+1}\Phi_{j+1} = Q_j; \qquad j = 0,1,2,\ldots; \tag{11}$$

$$\Phi_{j+1} = [\phi^1_{j+1},\ldots,\phi^G_{j+1}]^T; \quad Q_j = [S^1_{f,j},\ldots,S^G_{f,j}]^T;$$

Therefore, for a given vector Φ_j the algebraic system (11) is solved to obtain the neutron fluxes Φ_{j+1}. Hence, the general process requires for the first time-step an initial flux vector which is used in (11) to determine the new neutron fluxes at the end of the time step, thus using these new ones to calculate the new delayed neutron precursor concentrations vector. This process is carried out for each time step over the total time interval $(0,T]$.

3. AZKIND algebraic system

As result from the application of the NFE method RTN-0 and the use of the θ-method approach, the corresponding algebraic system is represented as a matrix–vector multiplication $A\vec{v} = \vec{b}$, where A and $\vec{b}$ are known and the unknown vector $\vec{v}$ is computed for each time step Δt_l.

The square matrix A is non-symmetric and it is comprised of G^2 matricial blocks for G neutron energy groups. Each block is composed by a tridiagonal set of elements and a set of sparse elements. Sparse elements correspond to the numerical coupling between the average nodal flux with each of the neutron fluxes at the edges of the node. Each block is built up with matrix elements corresponding to the directions x, y, z, and c.

The algebraic system to be solved in AZKIND is a difficult problem when it is required to compute the solution for real nuclear reactor cores for a considerable number of G neutron energy groups. It is difficult in the sense that the non-symmetric matrix A becomes a very large matrix which changes for each time step. The matrix generated by AZKIND has only *nnz* non-zero elements (see Section 6.1).

Searching for an iterative algorithm for non-symmetric linear systems, the method bi-conjugate gradient stabilized (BiCGStab) (Van der Vorst , 1992) was used because it is demonstrated to be an efficient and fast converging method for Krylov subspaces.

4. The AZKIND code and parallel computing

4.1. AZKIND background

The computer code *AZtlan KInetics in Neutron Diffusion* **AZKIND**, is part of the neutronic codes selected for implementation in the AZTLAN Platform (Gómez-Torres et al., 2015). The original development NRKin3D was a master degree project (Rodríguez-Hernández, 2002) written in FORTRAN 77. The first task in converting this academic code to a real reactor simulator code was to migrate to FORTRAN 90/95, mainly for the dynamic management of memory allocations. The next step was the implementation of algorithms for parallel processing to generate results in short acceptable times. AZKIND is comprised of the computer subprograms PRTN0 (Salas-Cuevas, 1995) and NRKin3D which calculate the nuclear reactor steady state and the nuclear reactor kinetics, respectively.

4.2. Parallel computing library

PARALUTION (Lukarski, 2014) is an open source C++ library to solve sparse systems of linear equations with parallel processing. It offers a variety of iterative solvers such as the CG, BiCGStab and GMRES Krylov methods, and preconditioners based on additive (Jacobi, Gauss–Seidel) and multiplicative (ILUp, ILUT, power(q)-pattern enhanced multi-colored ILU(p) method) splitting schemes as well as approximate inverse preconditioning approaches, see Fig. 2.

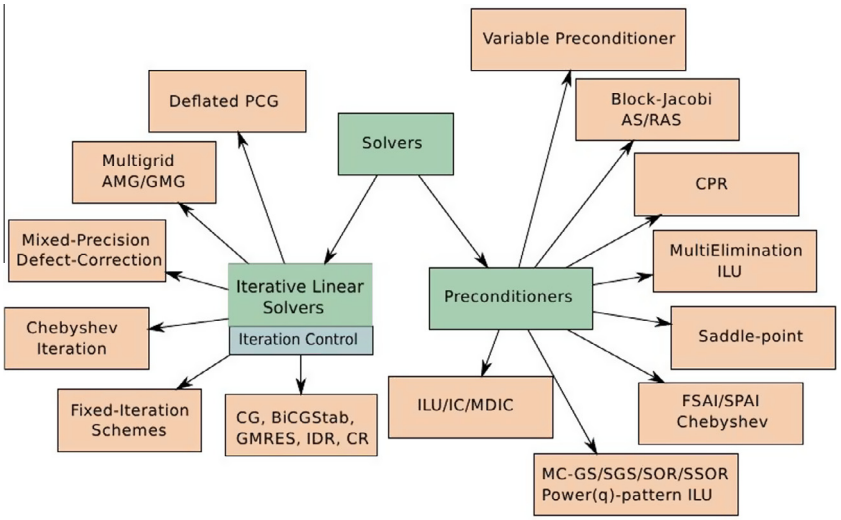


Fig. 2. Detail of the PARALUTION library architecture (Lukarski, 2014).

PARALUTION also features different matrix formats which are crucial for GPU (Graphics Processing Unit) internal bandwidth exploitation. Furthermore, it offers several hardware backends for execution on multi/manycore CPU and GPU devices. Due to its generic and flexible design, it allows seamless integration within any scientific software package, such as COBAYA3 (Trost et al., 2014) and the present AZKIND project (Gómez-Torres et al., 2015).

While there are several scientific libraries for our purpose of speeding up the computation time of the sparse linear systems solution procedure, such as ViennaCL (Rupp et al., 2010), Intel MKL (Intel Math Kernel Library (MKL), 2016), AMD Core Math Library (Core Math Library (ACML), 2016), Nvidia cuSPARSE (NVIDIA cuSPARSE Library, 2014), or the new release of NVIDIA OpenACC Toolkit (NVIDIA OpenACC Library, 2015), only the PARALUTION library features the high portability of both, code and obtained results, on modern state of the art hardware as one of its key features.

Nowadays, multi-core CPUs (OpenMP, MKL), CUDA supporting GPUs and OpenCL capable hardware (Intel Xeon Phi, AMD GPUs) support PARALUTION library. This offers the possibility to switch between different architectures without modifying any line of existing code and thus exploiting the available computational power of almost any computer system. This implies that software developers do not necessarily have to deal with architecture-related programming models.

PARALUTION comes with a plug-in support for FORTRAN, OpenFOAM and Deal.II packages. The FORTRAN plug-in makes possible to communicate FORTRAN- compiled libraries with the C++-compiled library with the use of an appropriate interface module. The FORTRAN plug-in was integrated in the specific project AZKIND to be run in different architectures.

4.3. Parallel processing implementation

Given a particular reactor core configuration, the AZKIND code has the option of simulating a steady state (SS) nuclear reactor condition, a dynamic case using previously generated initial steady state data as an input, or simulating a steady state plus a nuclear reactor dynamics calculation. The diagram in Fig. 3 shows how AZKIND has been constructed including the implementation of the PARALUTION FORTRAN plug-in.

On the left side of Fig. 3, a dynamic calculation is performed using the PARALUTION solver called in AZKIND with the use of the interface module. The AZKIND flowpath is shown in the right side.

The dynamic calculation is performed sequentially for each time step, until the total interval $(0, T]$ is completed. Here, it is important to point out that in the original AZKIND code the Bi-CGStab solver was implemented using the reference (Van der Vorst , 1992), working in a serial computational process. The preconditioning matrix was constructed with the inverse of the square root of each diagonal matrix element, $(\sqrt{a_{ii}})^{-1}$. Section 6.2 contains brief details about a verification task to prove the good functionality of parallel process implementation.

AZKIND works with vector arrays for each x, y, z, and c direction (see Fig. 1), while PARALUTION receives full arrays only. Then, first of all, it was necessary to create a subroutine to write the AZKIND matrix A in a Matrix Market format (Matrix Market, 2016), ready to be read by the PARALUTION solver. The first implementation of the FORTRAN plug-in was writing the matrix in Matrix Market format in a file allocated in the hard disk and the procedure in AZKIND opened the file and read the matrix from the file to be assigned to an array in AZKIND an to be transferred to PARALUTION via the interface module between FORTRAN and C/C++. The procedure was modified to eliminate the wasting time tasks of writing and reading in a file. Now the matrix coefficients are computed and allocated in the corresponding array (computer's memory) that is transferred to the PARALUTION solver via the module interface.

Also, the *searched* solution vector $\vec{v}$ and the RHS vector $\vec{b}$ were constructed by concatenating the respective AZKIND directed-arrays, for each time-step before calling the PARALUTION solver.

In order to complete the algorithm on each time-step, it was necessary to create a subroutine to de-concatenate the reached solution vector $\vec{v}$ to have directed-arrays in order to continue the AZKIND procedure for each x, y, z, and c direction to update the neutron flux, recalculate the new delayed neutron precursors concentration, and to compute the new fission power rate; continuing this cycle until the time interval $(0, T]$ is completed.

The process with PARALUTION solver implemented is as follows: AZKIND reads the input data and generates A, initial $\vec{v}$, and RHS $\vec{b}$. PARALUTION FORTRAN plug-in reads the matrix in COO for-

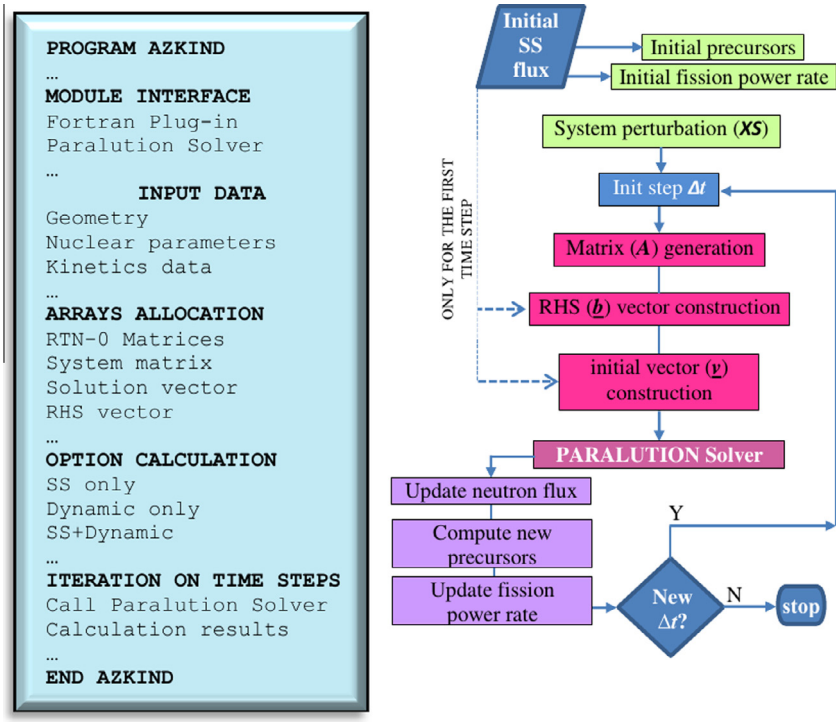


Fig. 3. AZKIND flowpath with PARALUTION parallel processing.

mat (Matrix Market, 2016), receives the vectors $\vec{v}$ and $\vec{b}$ and assigns the arrays memory. The plug-in calls the *paralution_solver* which is comprised of the ConvertTo() routine and the numerical solver bicgstab. The paralution_solver execution time accounts for these two processes running on GPUs. Before the execution of BiCGStab, the matrix COO format is converted to CSR format. In order to be able to perform direct solver execution time comparisons, the paralution_solver subroutine was temporarily modified to account only for the time spent by the numerical solver BiCGStab.

5. AZKIND for nuclear reactor steady state calculations

With the AZKIND code in a modern version some verifications were made using three nuclear fuel assemblies typical of a boiling water reactor (BWR). The nuclear data for the fuel lattices were generated with the Monte Carlo code SERPENT (Leppanen, 2007). These nuclear data for each lattice were supplied to AZKIND for nuclear calculations of the full assemblies.

Fig. 4 shows the axial (z) zones for the fuel assemblies where the numbers in each zone represent the material type for the respective lattice. In the right side appears the cross section (xy-plane) of the fuel assemblies. The assembly A is homogeneous for having the same material along the axial nodes, whereas the assemblies B and C are heterogeneous.

The xy-mesh used for calculation of the neutrons multiplication factor k_∞ was made by dividing x and y directions in two intervals. The z-mesh assumed was 25 axial nodes.

In the following table there is a comparison of the k_∞ values obtained with AZKIND and SERPENT:

The results generated with AZKIND are considered acceptable in comparison to those obtained with SERPENT. It is necessary to point out that the reason of the differences is because AZKIND is a neutron diffusion solver and it has been compared with a transport like solution obtained with Monte Carlo technique. See (Table 2).

6. High performance computing in AZKIND

6.1. Generation of matrices

Although the results with AZKIND for the three assemblies in Fig. 4 are acceptable, for simplicity in the construction of the input files for AZKIND, the basis was the homogeneous assembly A considering only material 1 along the 150 axial inches. This simplification does not reduce the computing challenge, since the computing complexity depends on the xyz-meshing.

The simplest case was considering only one assembly (array 1×1) where the xyz-mesh has 10 intervals in x-direction, 10 intervals in y-direction and 150 intervals in z-direction or axial direction.

Table 3 shows the matrix dimensions for different fuel assemblies arrays, where the largest case is the 10×10 array (100 assemblies), which can be considered as one-quarter of a small BWR reactor core.

In particular, for one-quarter reactor core, a smaller matrix could be used for a real reactor simulation if the z-mesh is changed from 150 to 25 axial nodes, as it is used in current BWR reactor core simulations. With this change the matrix dimension is 2,030,000 with 22,060,000 nnz elements; comparable to the 4×4 array dimension. Nevertheless, the fine z-mesh of 150 axial nodes was used in order to challenge the hardware capabilities and the performance of the AZKIND code with parallel computing.

6.2. Parallel versus serial results

The numerical solver used in AZKIND with serial process was an implementation, documented in Rodríguez-Hernández (2002), of the algorithm BiCGStab published in Van der Vorst(1992). The preconditioning matrix was constructed by multiplying the original main diagonals by the inverse of the square root of each diagonal matrix element; and as explained in Section 4.3, the parallel

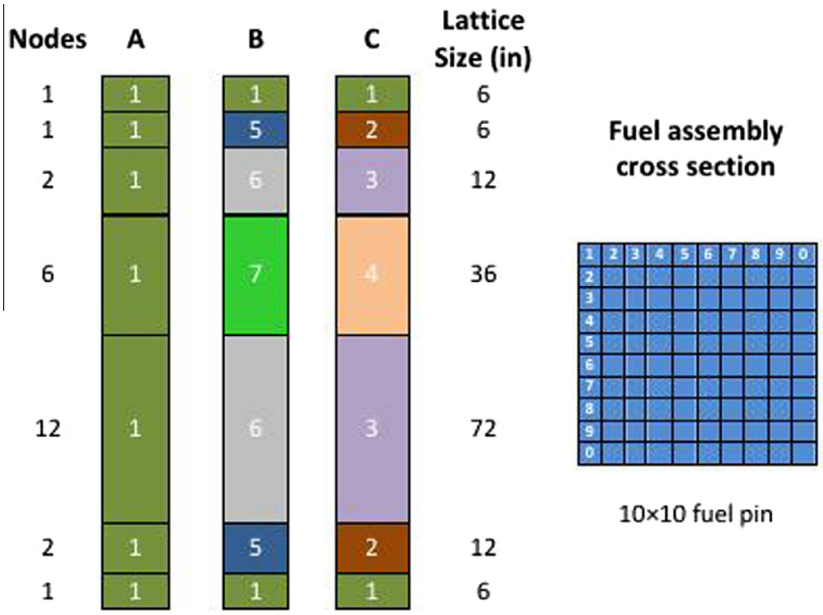


Fig. 4. Typical fuel assemblies for a BWR and a cross section view.

Table 2
Comparison of neutron multiplication factor k_∞.

Assembly	SERPENT	AZKIND	Error (pcm)
A	0.83541	0.83899	430
B	1.11991	1.11546	390
C	1.09758	1.09392	330

process in AZKIND is performed with the BiCGStab solver of the PARALUTION library, using the matrix preconditioner *MultiColoredILU*.

Three basic verification tests were used to verify independent results and consistency between the serial and parallel processes in AZKIND: (a) Perturbation null test; (b) Reproducibility test; and, (3) Infinite array test. Below there is a description of each test, together with a brief explanation of key issues solved in the parallel process implementation.

(a) *Perturbation null*. This test is to verify that the physical system (nuclear reactor core) remains absolutely without any change respect to the initial system condition, when a transient of duration T seconds is simulated with no perturbation at all. This test was successful in the serial module of AZKIND keeping the system power unchanged. However, when this test was applied to the parallel module some slight deviations were present because the system power was changing gradually: some adjustments were done in the use of double precision variables, and in the generation of the matrix that it is delivered to BiCGStab PARALUTION. The test was successful. The intent of this verification task is to reveal any error in the code running a transient without any change in the variables of the nuclear system: a steady state must be observed from the start to the end of the simulation time.

(b) *Reproducibility*. This test is to verify that for each particular matrix size the results with serial and parallel process are the same. A power transient of duration T is simulated for a perturbation introduced in the reactor system. In the first trials with the 1×1 case (one nuclear fuel assembly) some differences were noted from the 3rd to the 5th decimal place. Larger differences were generated for the 6×6 and the 10×10 fuel arrangements. After looking for some procedural difference between both computational processes, it was found that the absolute convergence criteria were different. Both criteria were set to 10^{-15} and the simulated transient results were equal in both computational processes; and as was expected, a significant difference in computing time was noted between serial and parallel processes (see Table 5 in Section 7).

(c) *Infinite array*. At the boundary of each array of assemblies the condition is to have neutron flux current equal to zero. This test is to verify that the power changes in a transient are equal for all the configurations, from 1×1 to 10×10 arrays of nuclear fuel assemblies. The result of this test was successful. Table 4 shows the results for the smallest an biggest arrays.

7. Numerical results

In the numerical experiments two types of simulations were performed. The first experiment was to observe the computational behavior of three kinds of GPUs by executing AZKIND with the different matrix sizes described in Section 6.1. The second experiment was to simulate a power transient in an actual reactor core with a fuel load of a boiling water reactor. The GPU cards used to run AZKIND with parallel processing were:

Table 3
Matrix dimensions for different fuel assembly arrays.

Assemblies	1×1	2×2	4×4	6×6	10×10
Matrix dimension	126,200	492,800	1,947,200	4,363,200	12,080,000
Non-zero elements	1,332,400	5,305,600	21,174,400	47,606,400	132,160,000

Table 4
Serial vs Parallel for the inifinite array test showing a power transient.

Time	1 × 1 serial	1 × 1 parallel	10 × 10 serial	10 × 10 parallel
0.1	1.000000	1.000000	1.000000	1.000000
0.2	1.066839	1.066839	1.066839	1.066839
0.3	1.148600	1.148600	1.148600	1.148600
0.4	1.244889	1.244889	1.244889	1.244889
0.5	1.359462	1.359462	1.359462	1.359462
0.6	1.497863	1.497863	1.497863	1.497863
0.7	1.515753	1.515753	1.515753	1.515753
0.8	1.522855	1.522855	1.522855	1.522855
0.9	1.529011	1.529011	1.529011	1.529011
1.0	1.535103	1.535104	1.535104	1.535104

1. GeForce GTX 860M with 640 processor cores and memory of 2 GB.
2. Tesla K20c with 2496 processor cores, and 4.8 GB of memory.
3. GeForce GTX TITAN X with 3072 processor cores, and 12 GB of memory.

7.1. Matrix sizes

Table 5 presents serial and parallel results for "one time-step". The AZKIND simulations were performed using the same power transient as it was used for the results in Table 4, i.e, 1 s divided in ten time-steps. Different solution times were observed for each time-step because as the power perturbation evolves, the algebraic system is different, giving different numerical behavior for each time step. Therefore, it was decided to get an average solving time for each array case in Table 5.

As already expected, the serial executions have very large times compared to the execution times of the parallel processing using the described equipment. The execution times in parallel processing also increase when matrix dimension increases, but these times are reduced when processors with more cores are used.

In Table 5, *No memory* means not enough memory in the GPU card to load very large matrices.

For the analysis of the computing acceleration or "speedup", a definition of speedup is used Nesmachnow (2015), known as relative speedup or speedup ratio:

$$S = \frac{T_1}{T_n},$$

where T_1 is the computing time using a single processor (serial calculation) and T_n is the computing time using n processor cores. The speedup achieved for each GPU using different matrix sizes is as follows,

Tables 5 and 6 show an excellent computing acceleration, despite of the speedup for small matrices that is comparable for the three computer architectures used. It also can be seen that the speedup values do not have a linear behavior. The non-linear behavior of the speedup is because although more GPU processor cores are used with massive data transference to and from the GPU, a data traffic delay is present in the communication bus between the GPU and the CPU.

7.2. Reactor power transient

Fig. 5 shows the distribution of nuclear fuel assemblies in the core of a boiling water reactor. The colors represent different types of fuel assemblies.

In the plane *xy* the mesh is 24 × 24, being coincident with each fuel zone; and axially, there is a partition with 25 intervals.

The matrix generated for this coarse mesh (1,274,304 *nnz*) is comparable to the matrix of the fine mesh created for the case of a unique assembly. See Table 5.

Before the simulation of a reactor power transient it was necessary to calculate the reactor steady state and compare the result with SERPENT. Table 7 shows the comparison of k_{eff} between AZKIND and SERPENT. The result for k_{eff} with AZKIND is quite acceptable, however the most useful result from this BWR steady state calculation is the neutron flux profile in the reactor core. This neutron flux is the departing point for a transient simulation.

Then, a power transient is simulated when the capability to remove neutrons is changed in the assembly identified as *perturbed* in Fig. 5. The neutrons removal capability is increased in the form of a step function during 3 s of the total power transient duration, which is 5 s, giving a reactor power reduction. The time step used in this simulation was 0.1 s.

Fig. 6 shows the power behavior over time, departing from a normalized value of 1.0 and reducing the power reactor almost to 80 percent of its original value.

This reactor power transient was simulated with the AZKIND code running on the three different GPUs listed at the beginning of this section.

Fig. 7 shows in logarithmic scale the time spent by AZKIND running in sequential mode (left bar) and the times spent by each GPU card.

As it can be seen in the logarithmic scale, serial processing is not competitive with parallel processing because it is demonstrated that serial computing employs lots of time. In Fig. 7, GTX refers to the GeForce GTX 860M device. In a comparison between GPUs, the GTX card with 640 processor cores employs the double of time compared to the Titan card with 3072 processor cores.

8. Conclusions and outlook

The most important issues derived from this work are summarized as follows:

With the aim of having a comprehensive implementation of parallel computing in AZKIND, the elements of the verification task are deemed to be useful to provide certainty that the results from serial and parallel simulations are mutually consistent and enough confidence was obtained in order to continue with high performance computing.

Considering a typical BWR nuclear fuel assembly it was possible to generate, from a single assembly up to large arrays of assemblies, increasing sizes of matrices for each case. In this sense, it was observed that parallel computing times are very small compared to the sequential computing, and that highly noticeable speedup is obtained using a high number of processor cores, mostly for large algebraic systems.

Table 5
Parallel processing time (seconds) in different architectures.

Assemblies	1 × 1	2 × 2	4 × 4	6 × 6	10 × 10
Matrix dimension	126,200	492,800	1,947,200	4,363,200	12,080,000
Non-zero elements	1,332,400	5,305,600	21,174,400	47,606,400	132,160,000
Serial	24	124	372	994	2471
GTX 860M	2.1	7.9	31.3	*No memory*	*No memory*
Tesla K20c	1.3	4.0	16.6	40.1	*No memory*
GTX TITAN X	1.0	2.6	10.4	26.7	95.4

Table 6
Speedup comparison (S).

Assemblies	1 × 1	2 × 2	4 × 4	6 × 6	10 × 10
GTX 860M	11	16	12	–	–
Tesla K20c	18	31	22	24	–
GTX TITAN X	24	48	36	37	26

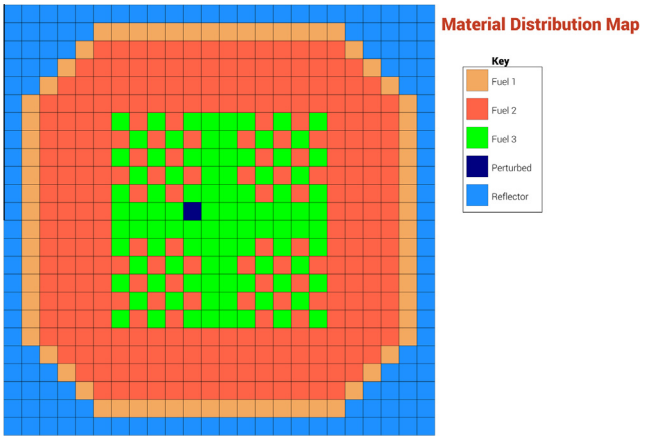


Fig. 5. Map of nuclear fuel assemblies in the core of a BWR.

Table 7
Comparison of neutron multiplication factor k_{eff}.

Case	SERPENT	AZKIND	Error (pcm)
BWR core	1.09472	1.09291	165

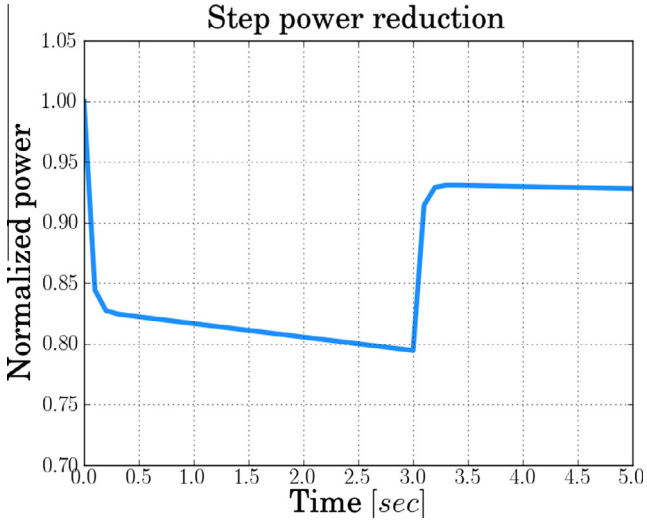


Fig. 6. Power transient profile for 5 s of simulation.

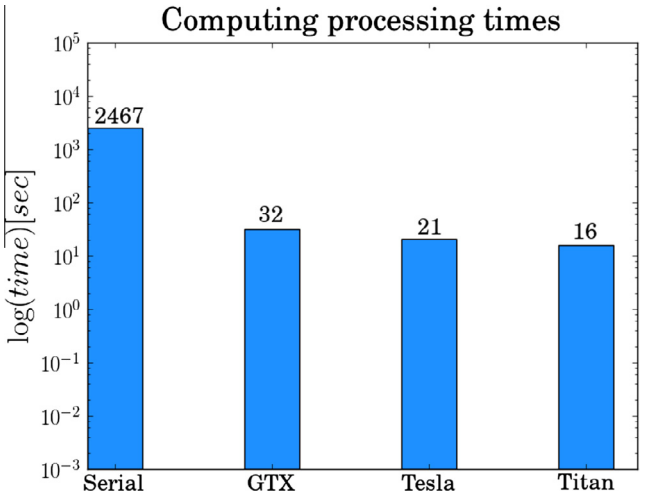


Fig. 7. Time consumption for serial and parallel processing.

The actual version of AZKIND has been demonstrated to be a computational basis with high computing capabilities to produce prompt results for real nuclear reactor calculations, as it was demonstrated in the power transient simulated.

The AZKIND neutronics code is envisaged to receive more developments to become a useful and powerful tool for the analysis of light water reactor cores with its strong capability to perform parallel computations.

In this work it is also demonstrated the key features of the PARALUTION library, like portability and scalability for its integration in specific FORTRAN projects such as AZKIND.

The experience gained from this work will be applied to implement parallel processing in other AZTLAN Platform codes: the neutron transport code AZTRAN, and the neutron diffusion code AZNHEX for hexagonal geometry. From the experience with AZKIND, the generation of the system matrix *A* is identified as a key issue due to its generation is a very specific task involving a significant effort.

Acknowledgement

The authors acknowledge the financial support (Gómez-Torres et al., 2015) from the National Strategic Project No. 212602 (AZTLAN Platform) as part of the Sectorial Fund for Energetic Sustainability CONACYT – SENER. Author M.Sc. Andres Rodriguez-Hernandez recognizes and appreciates the academical support

given by the Doctorate Program in Physics and Mathematics Sciences of the National Polytechnic Institute, Mexico (IPN-ESFM).

References

Duderstadt, J.J., Hamilton, L.J., 1976. Nuclear Reactor Analysis. Ed. John Wiley and Sons, New York.

Rodríguez-Hernández, Andrés, 2002. Solution of the Nuclear Reactor Kinetics Equations in 3D using the Nodal Method RTN-0 (Master Thesis). National Polytechnic Institute, ESFM, México.

Hennart, Jean P., 1986. A general family of nodal schemes. SIAM J. Sci. Stat. Comput. 1 (7), 264–287.

Grossman, Lawrence M., Hennart, Jean P., 2007. Nodal Diffusion Methods for space-time neutron kinetics. J. Prog. Nucl. Energy 49, 181–216.

Van der Vorst, H.A., n der Vorst(1992. BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonlinear systems. SIAM J. Sci. Stat. Comput. 13 (2).

Gómez-Torres, Armando M., Puente-Espel, Federico, Del-Valle-Gallegos, Edmundo, Francois-Lacouture, Juan L., Martin-delCampo, Cecilia, Espinosa-Paredes, Gilberto, 2015. Mexican platform for analysis and design of nuclear reactors. In: Proceedings of the International Congress on Advances in Nuclear Power Plants ICAPP.

Salas-Cuevas, Armando, 1995. Numerical Solution of the Neutrons Diffusion Equations, in XYZ using the Nodal Method RTN-0 (Master Thesis). National Polytechnic Institute, ESFM, México.

Lukarski, Dimitar, 2014. PARALUTION project, version 0.8.0. <http://www.paralution.com/> (accessed 2016-03-02).

Trost, Nico, Jiménez, Javier, Lukarski, Dimitar, Sanchez, Victor, 2014. Accelerating COBAYA3 on multi-core CPU and GPU systems using PARALUTION. Ann. Nucl. Energy 82, 252–259.

Rupp, K., Rudolf, F., Weinbub, J., 2010. Vienna CL A high level linear algebra library for GPUs and multi-core CPUs. In: International Workshop on GPUs and Scientific Applications, pp.51–56.

Intel Math Kernel Library (MKL), version 11.0, <http://software.intel.com/en-us/intel-mkl>, (accessed 2016-03-02)

Core Math Library (ACML), version 5.3.1, <http://developer.amd.com/tools-and-sdks/cpu-development/amd-core-math-library-acml>, (accessed 2016-03-02)

NVIDIA cuSPARSE Library, 2014. <https://developer.nvidia.com/cusparse> (accessed 2016–03-02)

NVIDIA OpenACC Library, 2015. <https://developer.nvidia.com/openacc> (accessed 2016–03-02)

Matrix Market, <http://math.nist.gov/MatrixMarket> (accessed 2016-03-02)

Leppanen, Jaakko, 2007. Development of a new Monte Carlo Reactor Physics Code (Ph.D. Thesis). Helsinki University of Technology, Finland.

Nesmachnow, S., 2015. Workshop Scientific computing on distributed memory systems. International Supercomputing Conference ISUM, Mexico.