Nuclear Mathematical and Computational Sciences: A Century in Review, A Century Anew
Gatlinburg, Tennessee, April 6-11, 2003, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2003)

IMPLEMENTATION OF MULTITHREADED COMPUTING IN THE MINOS NEUTRONICS FEM SOLVER OF THE CRONOS CODE

Jean C Ragusa
CEA-Saclay
Direction de l'Energie Nucléaire
Service d'Etudes des Réacteurs et de Modélisations Avancées
91 191 Gif sur Yvette, FRANCE
jean.ragusa@cea.fr

ABSTRACT

The MINOS solver of the FEM neutronics code CRONOS2 was parallelized using the directive based standard OpenMP. This implementation was tested on an SMP COMPAQ machine. The OpenMP implementation is only the first step towards fully using the SMPs cluster potential with a mixed mode parallelism. Mixed mode parallelism can be achieved by combining message passing interface between clusters withOpenMP implicit parallelism within a cluster.

Key Words: multithread parallelism, FEM MINOS solver, CRONOS code, SPN method

1. INTRODUCTION

The CRONOS2 code [1] was developed by the CEA for the design and the safety analysis of nuclear reactors. CRONOS2 is a 3-D Finite Element Method (FEM) code and solves various time-dependent approximations to the neutron transport equation (e.g. S_N, SP_N, diffusion) on structured grids (e.g. prismatic, hexagonal, and cylindrical geometries). The code has been extensively benchmarked, either by comparison with other equivalent codes --numerical benchmarking-- or by comparison with experimental results --experimental benchmarking: core start-up, core follow-up--. CRONOS2, as part of the SAPHYR system [2], is written in ESOPE/FORTRAN. The ESOPE language, developed by the CEA, is an extension of FORTRAN 77. ESOPE/GEMAT allows for dynamic memory management (dynamic allocation is performed once for all) [3].

Even on the most efficient workstations, the computational burden in CRONOS2 for solving the multigroup neutron fluxes--in steady or instationary state--on 3-D core geometries, can be considerable. The most time-consuming modules typically are: the matrices construction, the isotopic depletion, and the flux solver. With the generalization of shared memory symmetric multiprocessor (SMP) cluster architectures, many researchers combined the advantages of message passing computing with thread parallel computing [4,5,6]. In this mixed parallelism, clusters typically communicate through message broadcasting interfaces (such as MPI or PVM), whereas highly implicit parallelism is used within a cluster through multithreading (with, for instance, OpenMP or Posix threads). Previous work using MPI was successfully tested in an older and no longer available version of the CRONOS code, on a Cray T3D machine [13].

The purpose of this paper is to present the parallelization of the mixed dual FEM flux solver MINOS using thread parallelism (OpenMP [7]). This paper is organized as follows: the mixed dual formulation, and its discretization, is presented in Sec. 2; in Sec. 3, the OpenMP implementation is described; some results and applications are given in Sec. 4.

2. THE MINOS FLUX SOLVER

Two transport solvers are mainly available in CRONOS2:

- the first one, PRIAM, uses a primal finite element approximation and solves the diffusion equations and the S_N equations on complex geometries (with two proposed variational principles: the even parity formulation and the even-odd parity formulation) [8,9]
- the second one, MINOS, utilizes a mixed dual variational formulation and solves the diffusion equations and the Simplified P_N (SP_N) equations [10] on Cartesian geometries. By using the Raviart Thomas finite element basis, regularly shaped well-conditioned matrices are obtained. For the diffusion approximation, it was found that the MINOS solver performed 10 to 100 times faster than the PRIAM solver, depending on the problem properties and the computer hardware [11].

2.1. SPN Equations

The SP_N equations, with anisotropic scattering, are written on a given domain Ω as follows:

$$\begin{aligned} &\ell\vec{\nabla}\psi_{e-1}+\tilde{\sigma}_{ae}\psi_e+(e+1)\vec{\nabla}\psi_{e+1}=\vec{S}_e \qquad \text{for } e \text{ even}\\ &\ell\vec{\nabla}.\vec{\psi}_{o-1}+\tilde{\sigma}_{ao}\psi_o+(o+1)\vec{\nabla}.\vec{\psi}_{o+1}=S_o \qquad \text{for o odd}\end{aligned} \tag{1}$$

where $\tilde{\sigma}_{a\ell}=(2\ell+1)\sigma_t-\tilde{\sigma}_{s\ell}$

$$S_\ell=\sum_{g'\neq g}\Sigma_{s,\ell}^{g'g}\psi_\ell^{g'}+\delta_{\ell 0}\frac{1}{I}\chi_g\sum_{g'}\nu\Sigma_f^{g'}\psi_0^{g'} \quad \text{or} \quad \vec{S}_\ell=\sum_{g'\neq g}\Sigma_{s,\ell}^{g'g}\vec{\psi}_\ell^{g'}$$

The boundary conditions on $\partial\Omega$ are written in a general fashion (Robin type or albedo) :

$$\psi_e=\sum_o \beta_{oe}(\vec{\psi}_o.\vec{n}) \tag{2}$$

2.2 Mixed Dual Variational Formulation

The variational principle is obtained by projecting the two equations (even on odd equations) on two different functional spaces (mixed formulation). The Green's formula is then applied to the odd equation (dual formulation). The mixed dual variational formulation is written as follows:

Find $(j,\vec{p})\in L^2(\Omega)\times H(\mathrm{div},\partial\Omega)$ such that :

$$\begin{aligned} &\int_\Omega\sum_o \alpha_{eo}(\vec{\nabla}.\vec{\psi}_o)j_e+\int_\Omega\tilde{\sigma}_{ae}\psi_e j_e=\int_\Omega S_e j_e=Q_e\\ &\int_\Omega\sum_e -\alpha_{oe}\psi_e(\vec{\nabla}.\vec{j}_o)+\int_{\partial\Omega}\sum_e \alpha_{oe}\psi_e(\vec{j}_o.\vec{n})+\int_\Omega\tilde{\sigma}_{ao}\vec{\psi}_o.\vec{j}_o=\int_\Omega\vec{S}_o.\vec{j}_o=\vec{Q}_o\end{aligned} \tag{3}$$

where $a_{ij} = id_{i-1j} + (i+1)d_{i+1j}$, with d_{ij} the Kronecker symbol, Ω is the domain problem and ∂Ω its boundary. Suffixes *e* and *o* stand for even and odd flux components (harmonics).

2.3. The matrix System

Raviart-Thomas-Nedelec (RTN) elements are excellent candidates for the approximation spaces, since resulting matrices are extremely sparse, with coupling terms only along the axes in rectangular geometries [12]. The global matrix system is formed as follows: the x, y and z components of the odd harmonics (vectorial unknowns) are numbered first; the even harmonics unknowns (scalar unknowns) are numbered last. The resulting system matrix for the even and odd fluxes can therefore be put in the following form (the zeroes in Eqs 4 are due to the choice of the RTN spaces, cf. Figure A1 in Appendix) :

$$\begin{bmatrix} -R_x & 0 & 0 & B_x \otimes H \\ 0 & -R_y & 0 & B_y \otimes H \\ 0 & 0 & -R_z & B_z \otimes H \\ {}^t(B_x \otimes H) & {}^t(B_y \otimes H) & {}^t(B_z \otimes H) & T \end{bmatrix} \begin{bmatrix} \overline{y}_o^x \\ \overline{y}_o^y \\ \overline{y}_o^z \\ y_e \end{bmatrix} = \begin{bmatrix} -\overline{Q}_o^x \\ -\overline{Q}_o^y \\ -\overline{Q}_o^z \\ Q_e \end{bmatrix} \tag{4}$$

Where $\mathrm{H} = [\alpha_{oe}]$ is the matrix coupling the angular harmonics (bi-diagonal matrix).

With a proper choice of polynomial basis for the even fluxes, the element matrix for the even unknowns can be diagonal; the global matrix T for the even unknowns is then diagonal, since, with the RTN element, scalar nodes are strictly interior nodes. The polynomial basis for the odd unknowns is chosen so that the coupling matrices B_d (d = x,y,z) are as sparse and simple as possible. With a judicious choice of basis, matrices B_d become a simple difference operator (i.e. a bi-diagonal matrix) and do not need to be stored. R_d matrices are block diagonal matrices (each block being constituted of a line of nodes belonging to the same line along direction *d*).

2.4. The iterative Algorithm

In order to obtain a positive definite matrix, the even flux is eliminated and substituted into the odd equations [10]. This elimination-substitution is straightforward, since *T* is diagonal. The solution algorithm is the same as the one developed for diffusion solver, the main difference being an added loop on the odd harmonics. The directional leakage terms associated with each odd component are used as the iterate vector. The iterative procedure is a block Gauss-Seidel algorithm, where each block is made of one of the x,y,z components of the odd flux vectors. The different stages of the iteration sweep are given below :

$$\left.\begin{aligned} X_d = Q_e - \sum_{d'<d} (J^{d'})^n - \sum_{d'>d} (J^{d'})^{n-1} = \left[Q_e - (F^d)^{n-1}\right] + (J^d)^{n-1} \\ W_d (\overline{y}_o^d)^n = Q_o^d + (B_d \otimes H) T^{-1} [X_d] \\ (J^d)^n = {}^t(B_d \otimes H)(\overline{y}_o^d)^n \\ \left[Q_e - (F^d)^{n-1}\right] = X_d - (J^d)^n \end{aligned}\right\} d := x, y, z \tag{5}$$

with : $W_d = R_d + (H^t H) \otimes (B_d T_d^{-1\,t} B_d)$ and $F^d = \sum_{d=x,y,z} J^d$

Note that in the first and last equations of the system 5 the quantities $Q_e - (F^d)^{n-1}$ have been introduced to indicate the use of the total leakage F as it is implemented in MINOS.
The W_d matrices have exactly the same block diagonal structure as the R_d matrices (since T is diagonal and B_d is bi-diagonal). Their inversion is performed using Cholesky decomposition (which is performed and stored before the flux solver routine is called). Solving the $W_d(\overline{y}_o^d)^n = \text{RHS}$ system can be done very efficiently on vector computers since each of the subsystems can be solved simultaneously (block diagonal structure, cf. Figure A2 in Appendix). At the end of the process, the even flux is updated and is used to compute a new source.

3. IMPLEMENTATION OF OPENMP ON THE COMPAQ SMP CLUSTER

With the increasing numbers of SMP clusters, mixed mode parallelism is a requirement for modern, state-of-the-art neutronics codes, such as CRONOS2. Our work reported here only tackles the multithread parallel aspect of mixed mode parallelism, which we hope to study in a near future. Our conclusions will focus again on that aspect.
OpenMP is one of the major techniques used for multithreaded computing. OpenMP is an ensemble of directives with specific clauses and their associated libraries. It works on the fork and join concept of creating a team of threads at the beginning of each parallel region and deleting the team, excepting the master thread, on leaving a parallel region.
Creating several parallel regions within a program or subroutine makes its implementation much easier, since concerns regarding the local and global range of variables is greatly alleviated, but the overhead of creating and deleting a team of threads could result a very minimal parallel performances, or speed-up. This can be the case when parallel regions are imbedded within loops.

The computer used for our tests is the CEA-COMPAQ SMP cluster "Ixia". It comprises 69 clusters, each one having 4 Digital EV68 processors (833 MHz, 1.2 ns cycle time). OpenMP can be utilized on a cluster node, by starting up to 4 threads efficiently, since there are 4 processors per node.

The MINOS algorithm utilizes mainly basic linear algebra routines (cf. Figure 3 in Appendix for the algorithm flowchart):

- most of the computation is performed within the loop on the directions, which is included in the inner iterations loop
  - updating the various leakage vectors requires difference, addition and copy, SAXPY, and index permutation subroutines (the index permutation is needed for vectorization).
  - solving the W_d systems requires matrix-vector multiplication and vector difference subroutines
- the energy loop involves
  - fission source updates (matrix-vector multiplication)
  - eigenvalue computation (scalar product)
  - convergence test (scalar product)

- possibly a Tchebychev acceleration (linear combination of vectors)

Table I provides the sequential code profiling edits regarding the problem described in Section 4.

Table I. MINOS solver profiling edits

Time, %	Time, Sec	Cumulated time, %	Cumulated time, Sec	Procedure (filename)	Procedure function
19.6	4.9775	19.6	4.98	aprod1_ (aprod1.f)	Matrix-vector multiplication
18.3	4.6279	37.9	9.61	scopy_ (scopy.f)	Vector copy with index permutation
17.9	4.5283	55.8	14.13	cdif_ (cdif.f)	Vector difference
17.0	4.3193	72.8	18.45	cdimu2_ (cdimu2.f)	Vector multiplication and difference
6.5	1.6523	79.3	20.11	cdsdot_ (cdsdot.f)	Scalar product
5.0	1.2676	84.4	21.37	cdifv_ (cdifv.f)	Vector difference
4.4	1.1104	88.7	22.48	caddv_ (caddv.f)	Vector addition
3.4	0.8555	92.1	23.34	aprad1_ (aprad1.f)	Matrix-vector multiplication
2.7	0.6777	94.8	24.02	cadd_ (cadd.f)	Vector addition
1.0	0.2480	95.8	24.26	inir_ (inir.f)	Vector initialization

4. RESULTS

Our first attempt at parallelizing MINOS with OpenMP was to start and end a parallel region within each BLAS subroutines. This resulted in a serious performance loss, since the overhead in creating and deleting a team of threads within loops completely override the parallel gain. Since the code is written in FORTRAN/ESOPE, starting a parallel region where FORTRAN and ESOPE co-exist rose the problem of compability between ESOPE and OpenMP. In a FORTRAN/ESOPE code, a single FORTRAN COMMON is allocated and memory management is performed within that COMMON. An ESOPE structure is similar to a C language structure and contains work-arrays called segments. A segment's first element position is given by the integer pointer.

In order to start only one parallel region within the flux solver, a unique parallel region was created just before entering the outer iterations loop. At that point within the solver, all ESOPE structures are already activated, and temporary segments and pointers are already created. Nevertheless, many segments are only activated when needed within the loops. This segment activation within the parallel region leads to shared/private conflicts. The MINOS solver is then modified to allow for full memory activation prior to the outer iterations loop. The global ESOPE COMMON is unequivocally shared by all threads at all times, which greatly simplifies the OpenMP implementation. The entire outer iteration loop was declared as a parallel region in the OpenMP definition. All subsequent BLAS subroutines are parallelized with the OMP DO directive. Some attention is required for a few variables, such as explicit increment. Also, the

permutation subroutines need to be re-arranged to allow for the use of the OMP DO directive. All in all, only six OMP SINGLE regions have to be declared within the outer iterations loop, minimizing the implicit BARRIER and FLUSH operations at the OMP END SINGLE directive.

The performance of the OpenMP implementation in the MINOS solver is analyzed on the following practical case:

- a 900 MWe PWR core, 157 fuel assemblies (including 32 MOX assemblies), 3-D geometry, assembly homogeneous 2-group cross-sections.

A quadratic polynomial basis in directions X,Y and Z is chosen for all computations. Gauss-Legendre numerical integration formula is used. The initial computational mesh (referred to as mesh H in Table II, consists of 20 planes in the Z-direction and 1 square mesh per assembly in the XY-direction. The computational mesh is refined in the X, Y and Z directions to evaluate the speed-up dependence on the problem size. Table II summarizes the speed-ups achieved in the OpenMP parallelization of the MINOS flux solver. It is noticed that, for a rather small number of unknowns, speed-ups achived fall under 2. There is a trade-off between starting threads, synchronizing them, and the size for the loops that are to be computed in parallel. When a critical problem size is reached, speed-ups increase slowly with the problem size, since more time is spent in parallel loops.

Table II. Speed-ups obtained using 4 threads on the SMP COMPAQ Machine (OpenMP code)

Mesh refinement	# of odd unkowns (per energy group)	Time in Sec, 1 thread	Time in Sec, 4 threads	Speed-up
H	~44 000	14,83	9,31	**1,59**
H/2	~1 055 000	106,73	49,14	**2,17**
H/4	~3 558 000	391,08	178,11	**2,20**
H/8	~8 434 000	1086,62	477,92	**2,27**

5. CONCLUSIONS

Shared memory architectures are gradually becoming more prominent in the high performance computing field, since larger numbers of CPUs are allowed to access a single memory space. Moreover, these SMP systems are also clustered together to go beyond the performance of a single system (be it in terms of memory allocation or for reduced execution times). It is of prime importance for the CEA's applications to be portable and efficient on clustered SMPs. A more powerful COMPAQ SMP cluster is scheduled to enter production by the end of 2003, and a Linux PC farms, constituted of Pentium-IV bi-processors, is planned for early 2003 at the SERMA (Service d'Etudes des Réacteurs et de Modélisations Avancées).

In our work presented here, the directive based OpenMP parallelism was implemented in the FEM MINOS solver of CRONOS2 and tested on an SMP COMPAQ machine using 4 threads. We plan to extend the OpenMP implementation to other modules of the CRONOS2 code, such a the matrix computation module or the isotopic depletion module; higher speed-ups

are also expected on these modules, since each computational mesh can be treated independently in most of these modules. As a second stage, we hope to develop a mixed mode MPI / OpenMP CRONOS2 code.

MPI / OpenMP mixed mode codes could potentially offer the most effective parallelization strategy for an SMP cluster. Previous work using MPI was successfully tested in an older version of the CRONOS2 code, on a Cray T3D machine [13]. In that work, one or several W_d blocks were assigned to each processor. For mixed mode parallelism, there is no apriori obstacle in assigning several W_d blocks on a cluster, and then, within a cluster, using OpenMP as the inner level of parallelism.

REFERENCES

1. J.-J. Lautard, S. Loubière, C. Fedon-Magnaud, "CRONOS: a Modular Computational System for Neutronic Core Calculations", *IAEA Specialists Meeting on Advanced Calculational Methods for Power Reactors*, Sept. 1990, Cadarache, France
2. C. Magnaud et al., "The SAPHYR code System", *Proceedings of PHYSOR 2002*, Seoul, Korea, Oct. 2002
3. P. Verpaux, "Manuel d'utilisation d'ESOPE, Version 10.0 et de GEMAT, Version 10.0", Rapport CEA - DMT/93.617 (1993)
4. I. Blush, C. Noble, R. Allan, "Mixed OpenMP and MPI for Parallel FORTRAN Applications", http://www.ukhec.ac.uk/publications/reports/ewomp_paper.pdf
5. D. Henty, "Performance of Hybrid Message Passing and Sahred Memory Parallelism for Discrete Element Modelling", Supercomputing, Dallas, 2000, http://www.sc2000.org/proceedings/techpapr/papers/pap154.pdf
6. Hybrid MPI / OpenMP Programming for the SDSC Teraflop System, Scientific Computing at NPACI (SCAN), http://www.npaci.edu/online/v3.14/SCAN.html
7. OpenMP, The OpenMP ARB, http://www.openmp.org
8. B. Akherraz, C. Fedon-Magnaud, J.J. Lautard , R. Sanchez, "Primal finite elements for the neutron transport equation with anisotropic scattering," *Nucl. Sci. Eng*. **120**, 187-198 (1995).
9. C. Fedon-Magnaud, "Pin-by-Pin Transport Calculation with CRONOS Reactor Code", *Proceeding of the ANS Mathematics and Computation International Conference*, Madrid, Spain, Sept. 1999
10. J.-J. Lautard, D. Schneider, A.-M. Baudron, "Mixed Dual Methods for Neutronic Reactor Core Calculations in the CRONOS System", *Proceeding of the ANS Mathematics and Computation International Conference*, Madrid, Spain, Sept. 1999
11. J.J. Lautard, "La méthode nodale de CRONOS : MINOS, Approximation par des éléments mixtes duaux," Note C.E.A-N- 2763. Août 1994
12. P.A. Raviart, J.P. Thomas, "A Mixed Finite Element Method for the 2nd elliptic problems, Mathematical Aspects of the Finite Element Method". *Lecture Notes in Mathematics* **606**, Springer Verlag (1977).
13. K. Pinchedez, "Calcul parallèle pour les équations de diffusion et de transport homogène en neutronique", Note C.E.A-N- 2844. Juillet 1999

APPENDIX

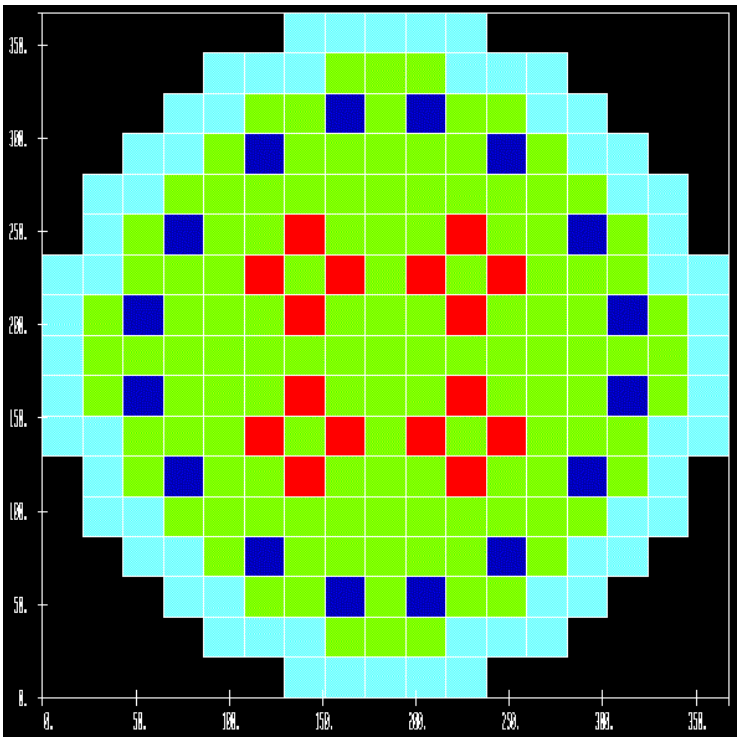
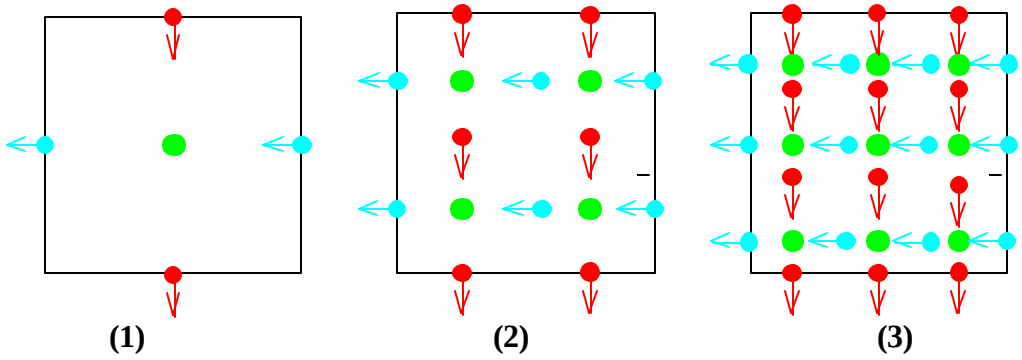


Figure A0. Core loading pattern (32 MOX assemblies)



(1) Linear : $\vec{p} \in P_{10} \times P_{01} f \in P_{00}$; **(2) Parabolic:** $\vec{p} \in P_{21} \times P_{12} f \in P_{11}$; **(3) Cubic:** $\vec{p} \in P_{32} \times P_{23} f \in P_{22}$

Even flux node, Odd flux node

Figure A1. The RTN element basis

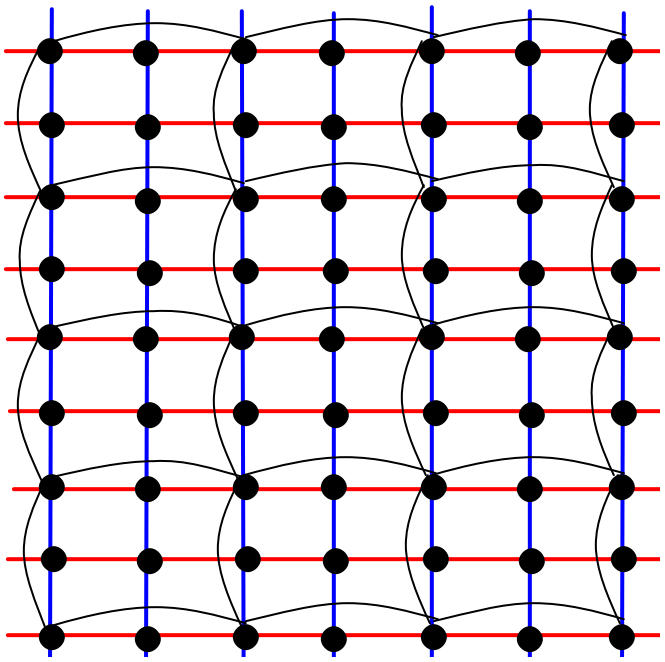


Figure A2. 2-D Example of current nodes coupling, with quadratic RTN element function

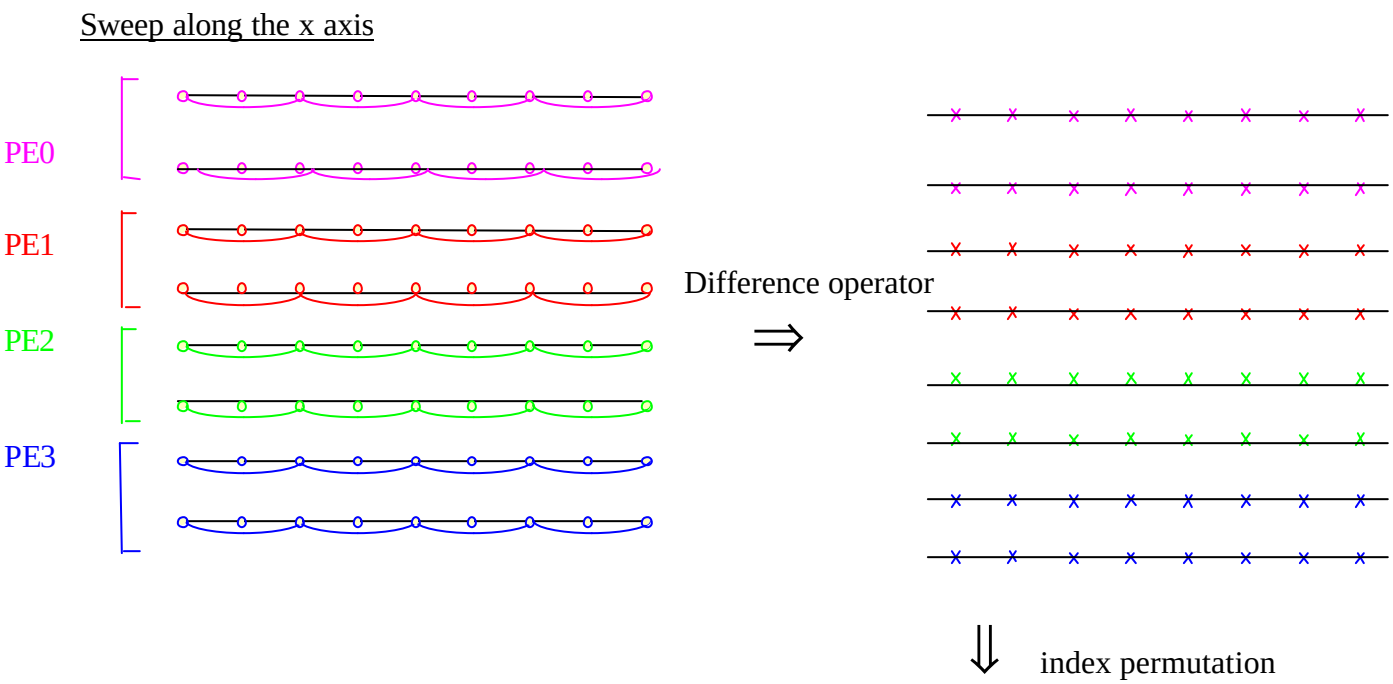


Figure A3. Schematics of the $W_d(\overline{y}_o^d)^n$ = RHS resolution, inner iterations loop (x,y example shown)

- Outer iterations Loop
  - Energy Loop
    - Sources computation (fission source, even and odd scattering sources)
      - Inner iterations Loop
        - Direction Loop
          - Compute X_d
          - Perform index permutation
          - Computed RHS $= Q_o^d + (B_d \otimes H) T^{-1}[X_d]$
          - Solve $W_d(\overline{\psi}_o^d)^n =$ RHS
          - Update X_d
        - End of Direction Loop
      - End of Inner iterations Loop
    - End of Energy Loop
    - New fission source
    - Eigenvalue computation
    - Convergence Check (if Yes, EXIT)
    - Tchebychev Acceleration
- End of Outer iterations Loop

Figure A4. MINOS algorithm flowchart