Optimization of the GPU-Based Depletion Solver in nTRACER

Han Gyu Lee and Han Gyu Joo

Seoul National University

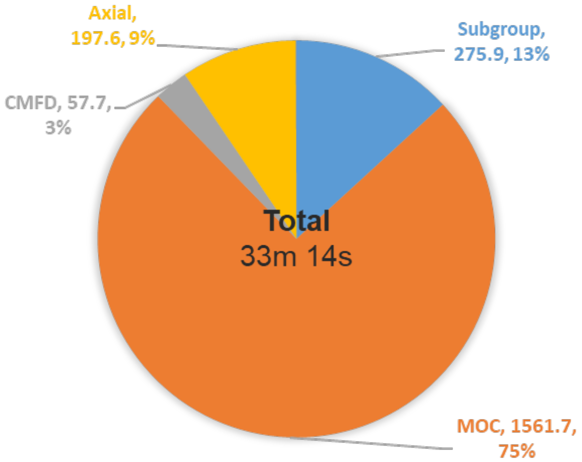
December 18, 2020

Contents

Status of the Initial Version of GPU Solvers in nTRACER

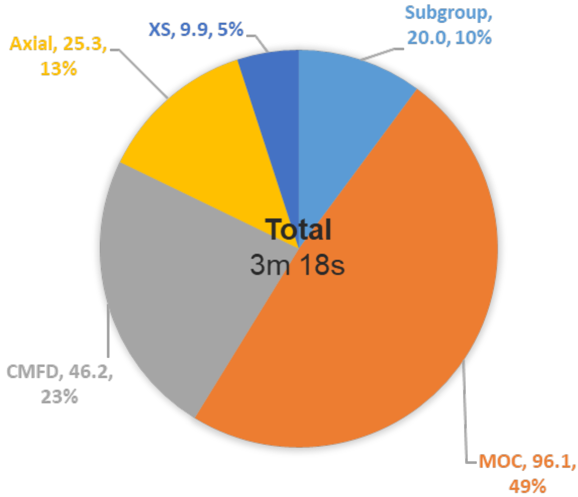
- **GPU Acceleration of nTRACER**
  - **GPU acceleration had been applied to the major parts.**
    - Method of characteristics
    - CMFD calculation
    - Axial solver
  - **The time for HZP problem decreased by a factor of 10.**
    - High speedup rate at MOC calculation

- **Degradation at Depletion Problem Calculation**
  - **The figure below shows time shares during 2D core depletion.**
  - **The time shares of the others were much higher.**
    - Cross section treatments
    - Subgroup
  - **This slowdown is because of increased number of nuclides and CPU dependence of the other modules.**
    - About 50% of the time was taken by cross section treatment.
  - **Thus, further exploitation of GPUs should have been applied.**

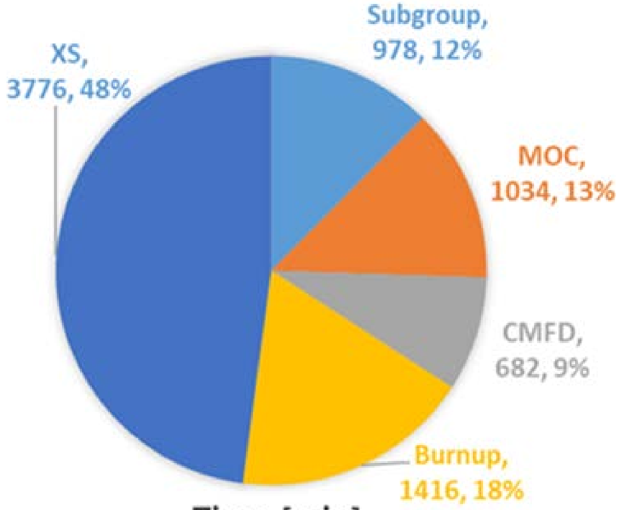


Xeon E5-2640 v3 256 Cores
(16 Nodes)

x 10



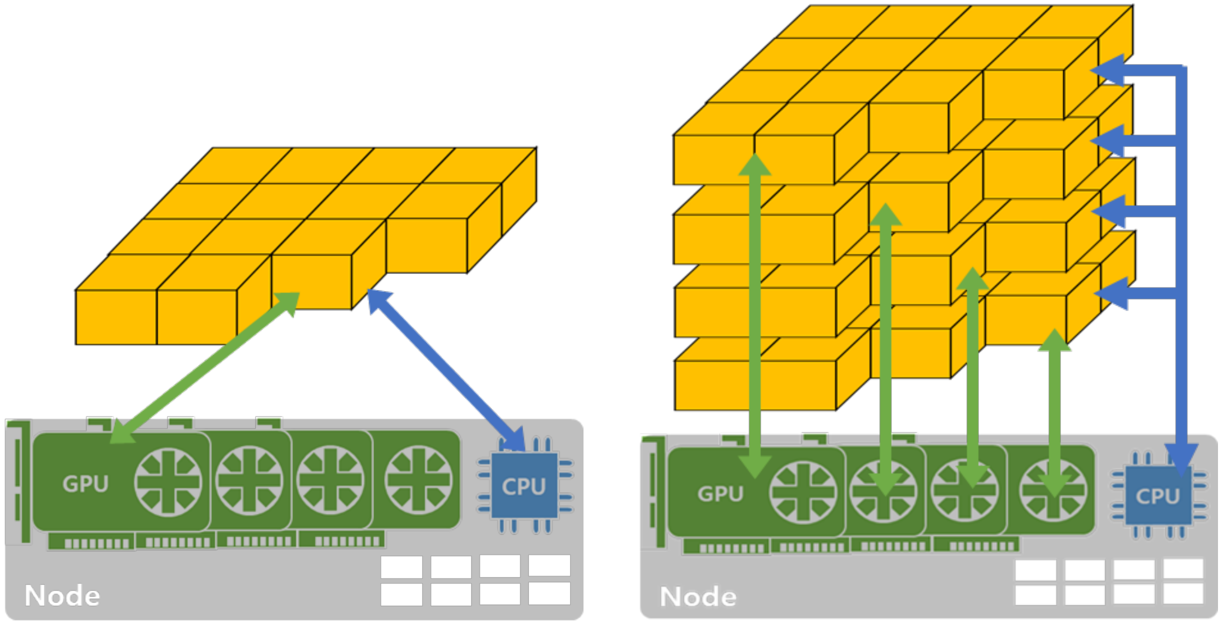
Xeon E5-2630 v4 80 Cores
+ 16 GeForce GTX 1080
(4 Nodes)



Time [min]

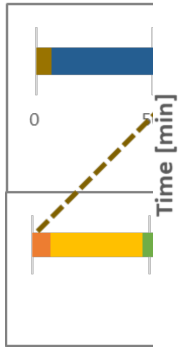
- **Topology of GPU Acceleration Modules in nTRACER**
  - **nTRACER assigns a plane to one GPU card.**
  - **Therefore, one node with multiple GPUs should treat many planes to fully exploit the GPUs.**
    - In general, heterogeneous systems have similar structure.
  - **In the meanwhile, the available CPU resources per plane are restricted for those cases.**
  - **This induces low scalability of 3D problems compared to 2D problems.**

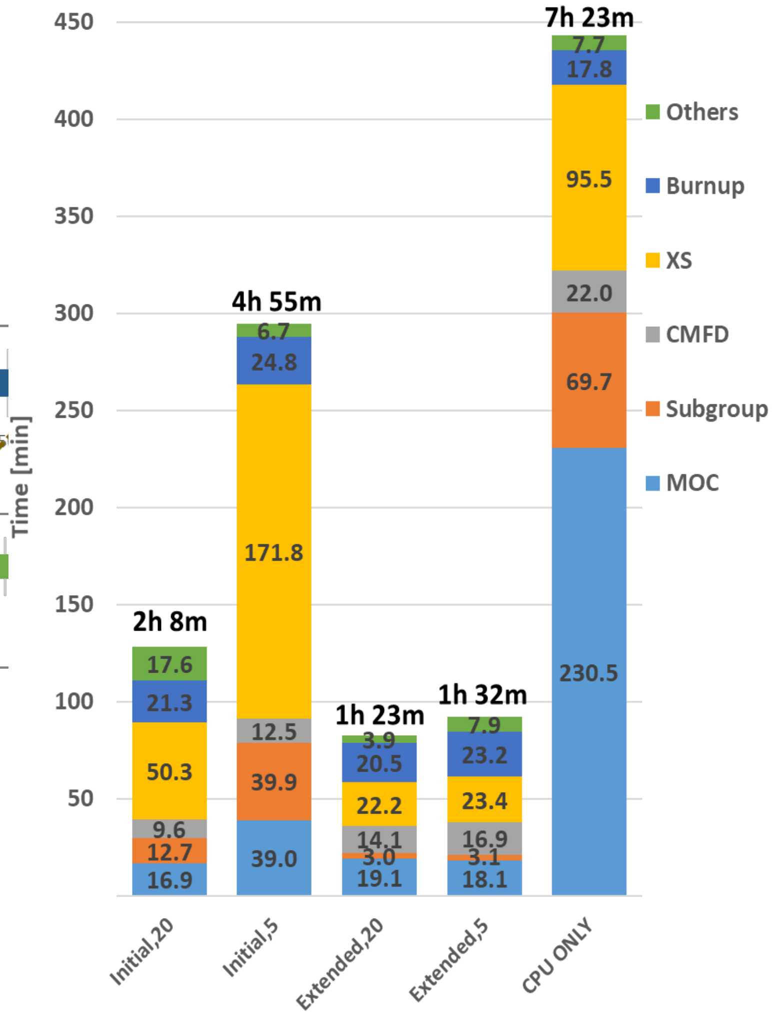
- **Exploitation of GPU to Lessen CPU Dependency**
  - **The major reason of poor scalability is high CPU dependency.**
    - Cross section treatments were all processed by CPUs.
  - **If fully utilizing GPUs for whole procedures, the performance at 2D problems can hold for 3D problems.**
  - **Thus, GPU acceleration should be extensively applied to the other parts.**
    - Depletion calculation
    - Cross section treatments
    - Minor processes contained by major procedures



Results of Extensive GPU Acceleration

- **Extensive Application of GPUs**
  - **The right figure illustrates performance for depletion problem with varying GPU usage.**
    - Initial : GPU modules only for major parts and depletion calculation
    - Extended : GPU modules for whole burdensome procedures including cross section
    - CPU ONLY : No use of GPUs
    - 20 and 5 indicates the number of CPU threads.
  - **The time of initial case was fine with 20 cores.**
  - **However, it became poor with fewer threads implying poor scalability.**
    - Even the MOC time increased.
  - **The extended cases show less difference while achieving better performance.**
    - Good scalability is promised.
    - Better performance of GPUs lessen the cross section time.





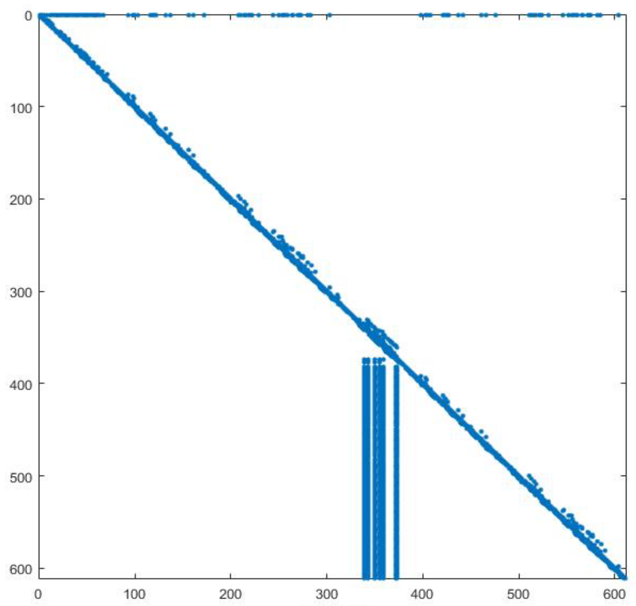
- **Inferiority of Depletion Solver**
  - **In spite of GPU porting of depletion calculation, it was slower than the CPU version.**
    - 2 or 5 minutes slower
  - **The performance improvements is the aim of works in this paper.**

Brief Descriptions about Depletion Solver

- Sparsity of Burnup Matrices
- GPU Acceleration of Depletion Solver

Sparsity of Burnup Matrices

- **Properties of Burnup Matrices in nTRACER**
  - **Construction of a burnup matrix is done with the fixed depletion chains.**
    - Given by the nTRACER depletion library
    - For all regions, the same chains are used.
  - **Due to the high sparsity, the matrices are stored with a sparse matrix format.**
    - Compressed Sparse Row (CSR)
    - Consists of a non-zero value vector, row pointer and column index vectors
  - **Thus, besides non-zero vectors, the two mapping arrays are identical for whole domains.**



- **Depletion System Batching**
  - **Temporal separation of tasks can be advantageous with the respect of parallel efficiency.**
    - Due to the locality of data
  - **The task separation requires a batch of systems of many regions.**
    - After construction of systems for domains of interest, matrix exponential problem is solved.
  - **Instead of an array of structures, the batch can be stored on a two-dimensional array.**
    - Thanks to the same number of non-zero elements of matrices
  - **The only data should be stored are non-zero elements, not index information.**

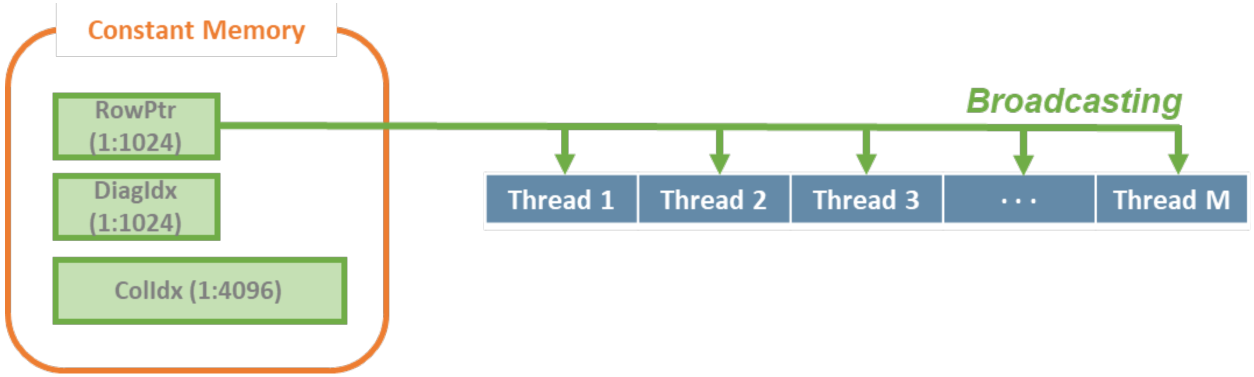
A(1,reg1)	· · ·	A(nz,reg1)
A(1,reg2)	· · ·	A(nz,reg2)
A(1,reg3)	· · ·	A(nz,reg3)
· · ·	· · ·	· · ·
A(1,regM)	· · ·	A(nz,regM)

- **Outline of the Initial GPU Depletion Solver**
  - **In the solver, each thread takes one region at a time.**
    - Region-wise parallelism
  - **It is based on Chebyshev rational approximate method (CRAM) with iterative solution.**
    - The red circled term is solved with iterative method.
    - BiCGSTAB is applied for the iterative solver.

$$\mathbf{n} = \alpha_0 \mathbf{n}_0 + \mathrm{Re}\left(\sum_k \alpha_k \left(\mathbf{A}\Delta t - \theta_k \mathbf{I} \right)^{-1} \mathbf{n}_0 \right)$$

$$\left(\mathbf{A}_k \Delta t - \theta_k \mathbf{I} \right) \mathbf{y} = \alpha_k \mathbf{n}_0$$

- **Sparsity Pattern Data on Constant Memory**
  - **It was pointed out that whole regions can share the same index arrays.**
  - **Those sparsity data can be transferred swiftly through use of constant memory.**
    - Constant memory is the special type of read-only memory that has fast access speed.
  - **Despite small size of constant memory, the sparsity pattern data are small enough to be stored on it.**

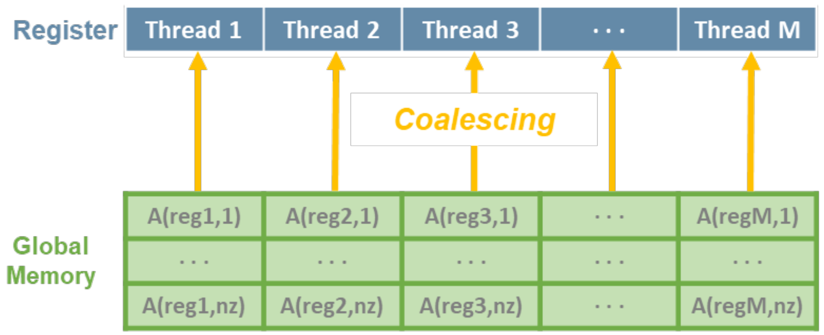


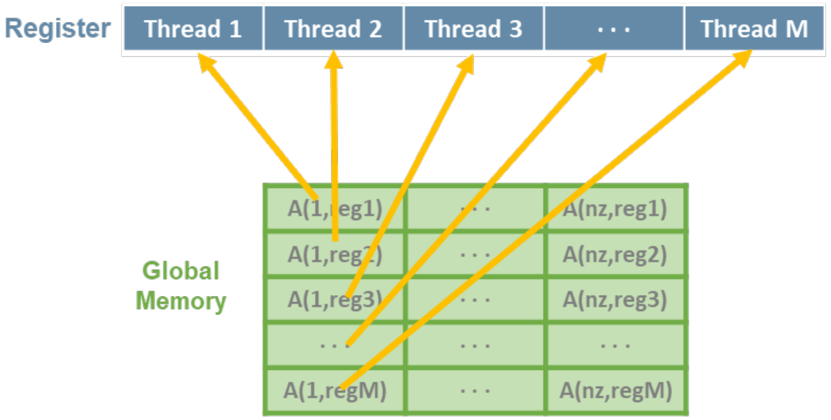
Optimization of Matrix Exponential Solver

- Non-Zero Element Major (NZEM) Storage
- Separating Diagonal and Off-Diagonal Elements
- Gauss-Seidel Iterative Solver

- **Conventional Storage of Depletion Systems**
  - **The matrix of each region is completely independent on the other regions.**
  - **This feature makes most of the solvers form the matrix arrays in region major (RM) order.**
    - The elements in a matrix are listed, which are followed by those of another matrix.
    - RM storage is a good option considering large size of L2 or L3 cache with a few or tens of CPU cores.

- **Advantage of NZEM Storage with GPU Architecture**
  - **Below two figures show the access pattern to system elements on global memory.**
  - **Under region-wise parallelism, thousands of threads access matrices at different regions.**
  - **With RM storage, elements having the same non-zero index are remotely located.**
  - **In the meanwhile, the NZEM storage enables coalesced memory access.**





Separating Diagonal and Off-diagonal Elements

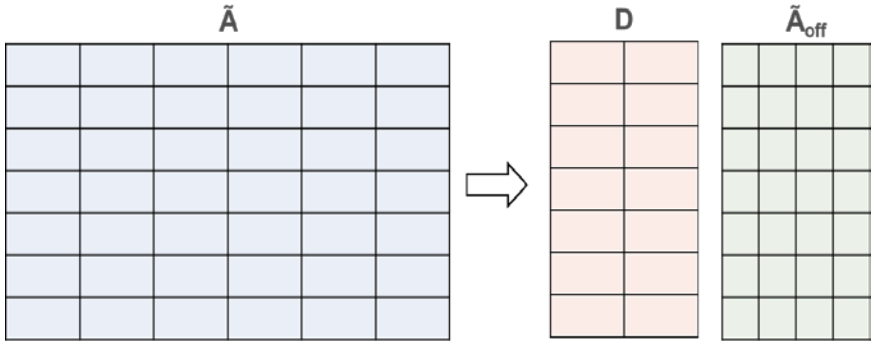
- **Need of Complex Number Variables**
  - **The formula in CRAM contains complex numbers.**
  - **The size of matrices may be doubled due to the complex type variables.**
  - **Not only for occupancy, the communication time increases as they are doubled.**
    - With confined bandwidth

$$\mathbf{n} = \alpha_0 \mathbf{n}_0 + \mathrm{Re}\left(\sum_k \alpha_k \left(\mathbf{A}\Delta t - \theta_k \mathbf{I} \right)^{-1} \mathbf{n}_0 \right)$$

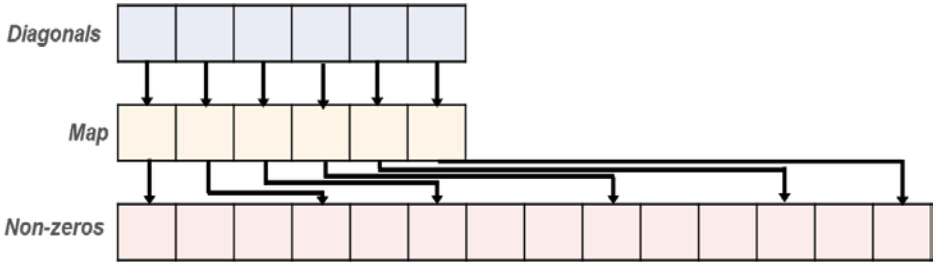
$$\mathbf{n}, \mathbf{A} \in \mathbb{R}^m,\ m : \text{the number of nuclides}$$

$$\alpha_j, \theta_j \in \mathbb{C}$$

- **Use of Real Type for Off-diagonal Data**
  - **The complex matrix, $\tilde{\mathbf{A}} \equiv \mathbf{A}\Delta t - \theta_k \mathbf{I}$, has complex numbers only on diagonal elements.**
  - **It is possible to separate the off-diagonals and store them in a real variable array.**
  - **The real array enables higher transfer rate of elements from global memory.**
  - **Also, the complexity of local operations is alleviated.**



- **Better Access Pattern to Diagonal Data**
  - **During iterative solution, it should read a diagonal element exclusively.**
    - Jacobi preconditioner with BiCGSTAB
    - Gauss-Seidel solution
  - **Reading only one diagonal in an indirect manner needs a mapping array.**
  - **However, the datum in separated arrays can be directly accessed.**



- **Disadvantages of the BiCGSTAB based Old Solver**
  - **Jacobi preconditioned BiCGSTAB was applied to the old iterative solver.**
  - **A BiCGSTAB solution requires much operations per iteration, but it is not problematic.**
    - If a kernel can fully utilize registers
  - **Large size of global memory should be allotted for independent buffers of residual and direction vectors.**
    - Vectors are also called as ‘workspace’.
    - At least 7 vectors per thread, while nTRACER assigns thousands of threads to a kernel
  - **Also, the access to them takes much of the solution time.**

- **Benefits from Application of Gauss-Seidel Method**
  - **The Gauss-Seidel based solver takes advantage from small iterations and buffers needed.**
    - Much simpler process than BiCGSTAB
    - Only 3 buffer vectors necessary
  - **Actually, it is turned out to take less iterations than BiCGSTAB.**
  - **Also, it is stable enough.**
    - For the more complex depletion problems, it is stable. (PRAGMA)

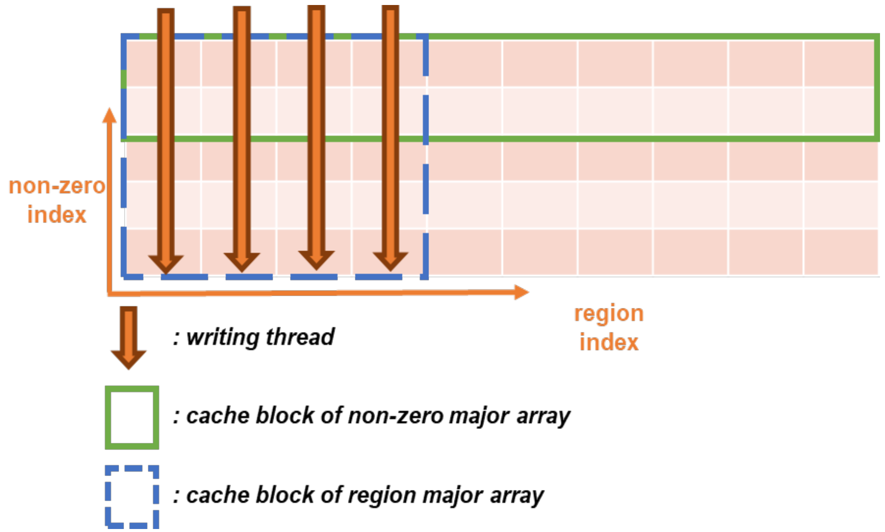
Overhead Optimization

- Implicit Transposition during Setup
- Fast Non-zero Index Search
- Explicit copy with CUDA API

ENGINEERING

- **System Array Transposition for NZEM Storage**
  - **As illustrated earlier, GPU achieves better performance with the NZEM ordering.**
  - **The setup routine used for CPU solvers returns an RM order array.**
  - **To copy NZEM array to the device, the array should be transposed.**
  - **However, a heavy bottleneck occurs during the processes of transposition.**
    - Typecasting and copy

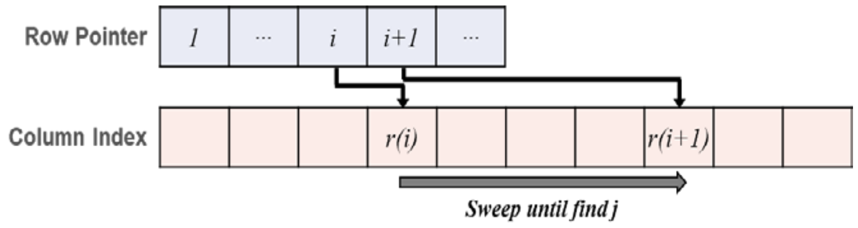
- **Implicit Transposition**
  - **To eliminate the overhead, the system data are directly filled in an NZEM array.**
    - Non-zero major off-diagonals and diagonals
  - **The matrix array in NZEM ordering is copied to a GPU without temporary copies.**
  - **In spite of no additional operations, this change increases the cost for setup.**
    - Smaller than the decrement of copy time
  - **The enlarged cost stems from low cache efficiency.**
    - Each thread takes the matrix of a region.
    - An RM array can hold data of the matrix on a cache.
    - However, the data of a region is not contiguously located in an NZEM array.
  - **At the expense of setup time, the temporary copy time can be reduced further.**



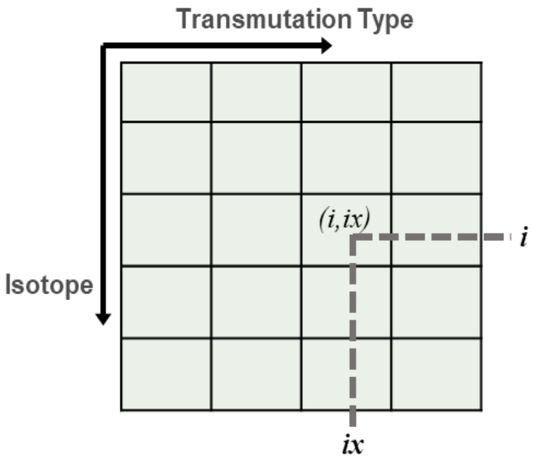
- **Indirect Accesses during Matrix Setup Phase**
  - **Most of sparse matrix formats require indirect access to an element.**
    - The row and column indices are listed in two vectors.
    - To locate a non-zero element, search through the index maps is needed.
  - **When adding transmutation rate, the proper non-zero index should be searched.**
  - **The search takes a few times of iterations while reading the map arrays.**
    - Converged use of bandwidth with multicores
  - **With limited use of CPU cores, this was not efficient resulting large time increment.**

$$\frac{dn_i}{dt} = -\left(\sigma_i \phi + \lambda_i\right) + \sum_j \sigma_{ji}\phi + \lambda_{ji}$$

$$\left(\mathbf{A}\right)_{ij} = -\delta_{ij}\left(\sigma_i \phi + \lambda_i\right) + \left(\sigma_{ji}\phi + \lambda_{ji}\right)$$



- **Tabularization of Non-zero Index with Nuclear Transmutation**
  - **Instead of search iterations, it is possible to tabularize the non-zero indices.**
    - A table contains the indices corresponding to transmutation types.
    - e.g. (n,γ), (n,2n), (fission), ···
  - **All the regions can share the single table due to an identical set of chains.**
    - All the matrices have the same sparsity pattern only varying reaction rates.
  - **This makes the location much faster with small number of cores.**



Explicit Copy with CUDA API

- **Overloaded Intrinsic of PGI Fortran to Device APIs**
  - **The PGI compiler has overloaded many intrinsic operations to handle device data.**
    - e.g. exp, sin, cos for variables in a kernel
    - allocate, deallocate of global memory on a device
  - **Exploiting it, the assignment operation (=) was used to copy-in the system data.**
  - **They are quite convenient, but sometimes, not effective for some cases.**

- **Use of cudaMemCpy Function**
  - **The same functionality can be done with cudaMemCpy function.**
    - Communication between host and device
  - **Thus, the assignment operators were all changed to them.**
  - **They shorten the time for communication with a factor of 5.**
    - The decrement is solely from communication between the host and device.
  - **As a result, the communication fully exploit the PCI-e bandwidth.**

Performance Analysis

- Problem Description
- Enhancements of Matrix Exponential Solver
- Reduction of the Overheads
- Overall Performance of GPU Depletion Solver

- **APR1400 2D Core Problem**
  - **Steady-state condition : HFP with infinite mass flow of coolant**
  - **Depletion condition : Up to 15 GWd/tHM**
    - Average step as 1 GWd/tHM
    - In sum, 18 burnup steps
  - **Depletion domain : ~ 74,000**

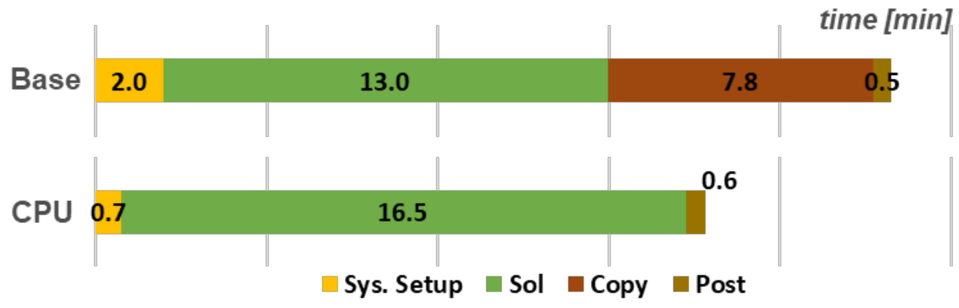
- **Computing Resource Specification**
  - **GPU version results : PGI Fortran 19.4**
  - **CPU version results : Intel Fortran 19.0.4**

CPU	2 x Intel Xeon E5-2630 v4 20 Cores, 2.4 GHz (Boost)
GPU	NVIDIA GeForce RTX 2080 Ti
Compiler	PGI Fortran 19.4 Intel Fortran 19.0.4

	J	K	L	M	N	P	R	S	T
9	A0	A0	C3	A0	B1	A0	B3	C2	B0
10	A0	B3	A0	B3	A0	B1	A0	B3	C0
11	C3	A0	C2	A0	C3	A0	C3	B1	B0
12	A0	B3	A0	B3	A0	B3	A0	B2	C0
13	B1	A0	C3	A0	C2	A0	B1	C0	
14	A0	B1	A0	B3	A0	B3	C1	C0	
15	B3	A0	C3	A0	B1	C1	C0		
16	C2	B3	B1	B2	C0	C0			
17	B0	C0	B0	C0					

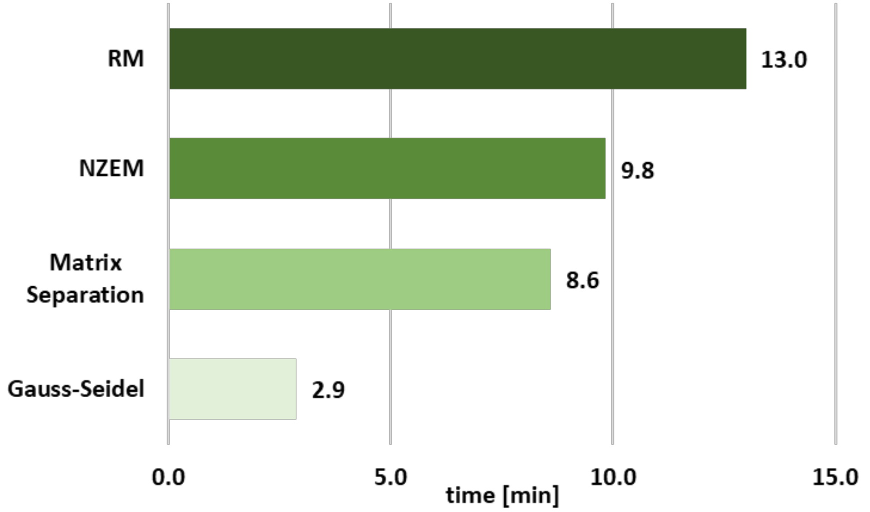
A0	1.72 wt% Fuel
B0	3.14 wt% Fuel
B1	3.14/2.65 wt% Fuel + 12 Gadolinia BP
B2	3.15/2.64 wt% Fuel + 12 Gadolinia BP
B3	3.14/2.64 wt% Fuel + 16 Gadolinia BP
C0	3.64/3.14 wt% Fuel
C1	3.64/3.14 wt% Fuel + 12 Gadolinia BP
C2	3.65/3.14 wt% Fuel + 16 Gadolinia BP
C3	3.64/3.14 wt% Fuel + 16 Gadolinia BP

- **Description of Items in Time Profiles**
  - **Sys. Setup : System setup time**
  - **Sol : Solution time**
    - Includes the execution of solver kernel and copy-out of solution vectors
  - **Copy : The time needed for copy-in to global memory**
    - Includes the copy-in and explicit transposition of the system
  - **Post : Post-processing time**

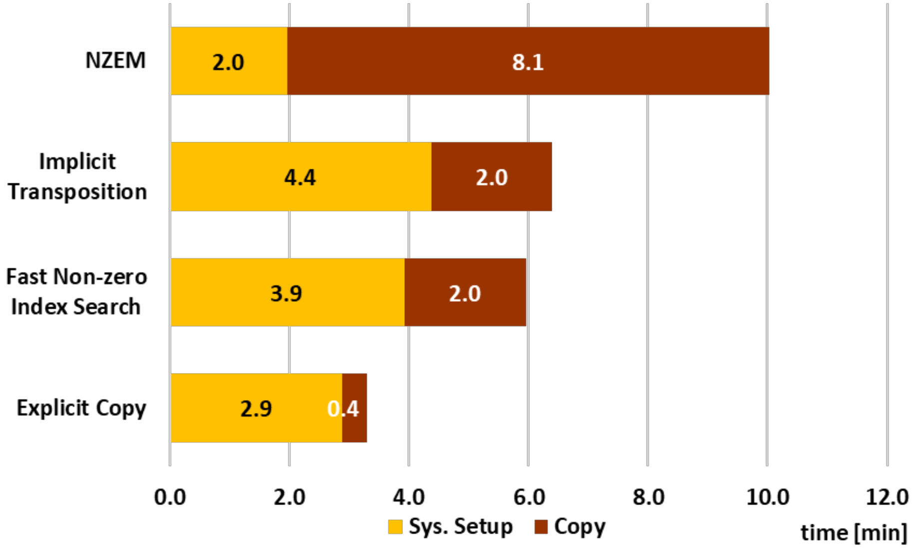


- **Poor Performance of the Old GPU Solver**
  - **The old GPU solver was even slower than the CPU version.**
  - **The 'Base' case shows better performance for the solution, but the others are largely increased.**
  - **Among them, setup time is slower due to the inferior performance of PGI compilers.**

- **Improvement of Solution Performance**
  - **Each case shows the improved solution time in a progressive manner.**
  - **The basic case, ‘RM’ takes 13 minutes for whole depletion steps.**
  - **By applying NZEM storage on the kernel, it shorten the time by 25%.**
    - The necessity of NZEM ordering
  - **Additionally, separation of matrices can decrease it further by 30% of the RM.**
  - **The replaced iterative method makes the most reduction of solution time.**
    - Thanks to much smaller workspace and less frequent global memory read

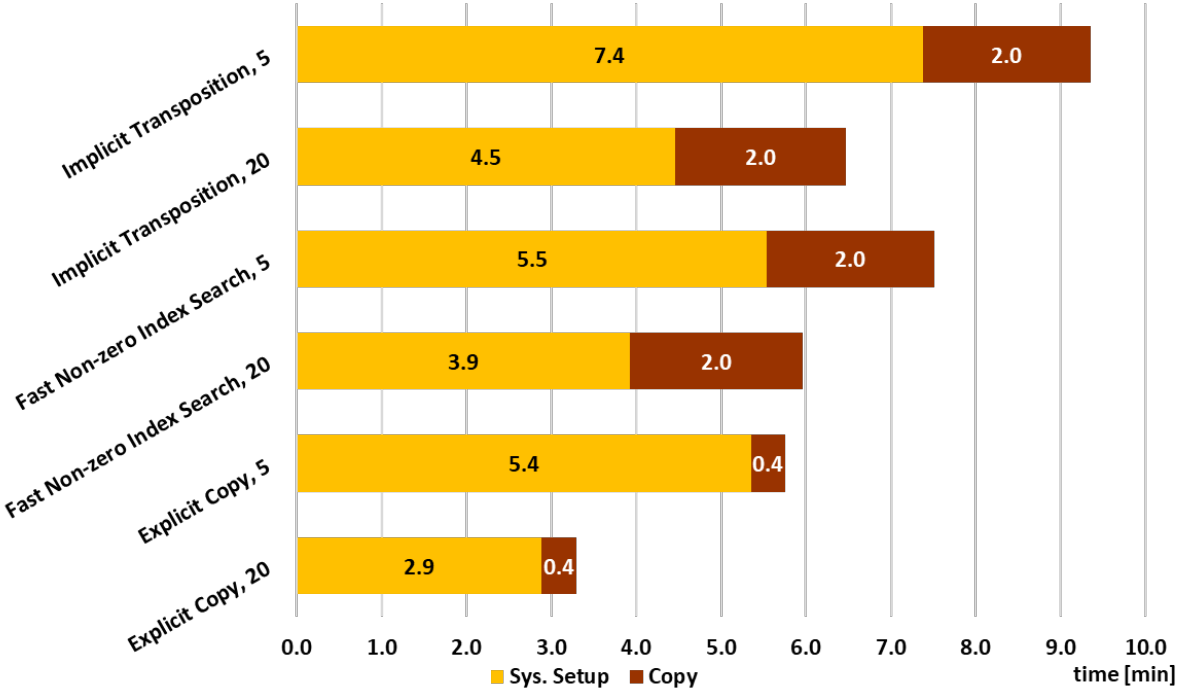


- **Overhead Improvements**
  - **The figure shows the time of two items, system setup and copy.**
    - The maximal number of cores were utilized for all cases.
  - **The largest portion at NZEM case was the copy time.**
    - Due to explicit transposition processes.
  - **Implicit transposition reduced it as 25%, while the setup time was doubled.**
  - **The benefit from fast index search was smaller than the other cases.**
    - It will be more evident with small number of CPU cores.
  - **Finally, explicit copy decreases the copy time under a minute.**

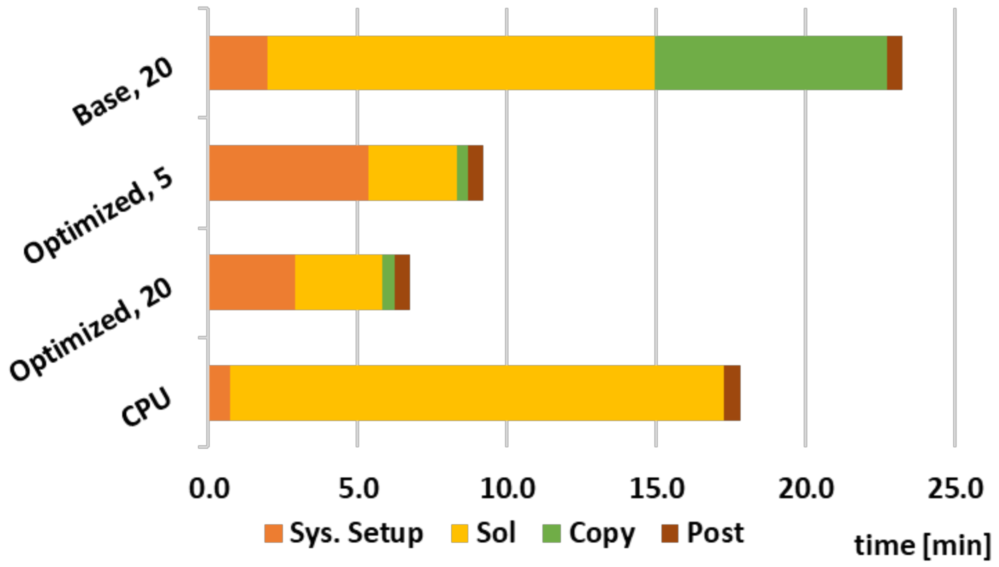


- **Overhead Profile with Limited Usage of CPU**
  - **Due to the topology of nTRACER, the performance with fewer CPU cores is an important parameter.**
    - When assigning the same number of planes as GPU cards in a node
    - Limited CPU resources is more realistic considering the 3D problems.
  - **The numbers at cases indicates the used number of CPU cores.**
    - 4 GPUs equipped per node
  - **This profile is analyzed to see the effect on setup time.**
    - The only part dependent on CPU resources

- **Effectiveness of Fast Index Search**
  - **Increment from limited CPU cores was found to be large.**
    - About 60% only applying implicit transposition
  - **The gap has been further decreased by fast index searches.**
  - **Also, the time with 5 cores decreases much from the fast search.**
    - 15% of decrease with 20 cores
    - Over 30% of decrease with 5 cores
  - **Explicit copy affects little on setup in the practical sense.**
    - Rather effective for the copy time



- **Overall Performance Comparison**
  - **Due to the overheads, the overall time of the old GPU solver was larger than the CPU solver.**
  - **After several times of optimization, the overheads have been decreased much.**
    - At the expense of setup time increment
  - **The matrix exponential solver was also optimized so that it is about one-fifth of the base case.**
  - **The speed of GPU solvers now definitely outrun that of the CPU version.**
  - **In addition, the CPU solver time would be degraded with small number of CPU cores.**
    - Above an hour with quarter of the full CPU resources



- **Completion of GPU Acceleration for Every Hotspot**
  - Including the optimized depletion solver, all the hotspots are successfully accelerated with GPU cards.
  - Now, every part has better performance compared to the CPU version.

- **NZEM Storage as GPU-friendly Data Structure**
  - NZEM storage has been proved to be efficient on GPUs' memory architecture.
  - By selection of compute-friendly data structures, 30% of the time has been decreased.

- **Effectiveness of Gauss-Seidel Based Iterative CRAM Solver**
  - It is stable enough with the simple iterative method.
  - In addition, the performance was very fast thanks to fewer buffers needed.

- **Overhead Reduction through Various Measures**
  - With the three measures, over 50% of the time from overheads were reduced.
  - Especially, under the short of CPU resources, they consumed only a few minutes.

- **Expectation of Good Scalability**