3D Cartesian Transport Sweep for Massively Parallel Architectures on top of PaRSEC

9th Scheduling for Large Scale Systems Workshop, Lyon

S. Moustafa, M. Faverge, L. Plagne, and P. Ramet
1

Context and goals
Guideline

Context and goals

Parallelization Strategies

Sweep Theoretical Model

DOMINO on top of PARSEC

Results

Conclusion and future works
Context

- EDF R&D is looking for a Fast Reference Solver
- PhD Student: Salli Moustafa
- Industrial solvers:
  - diffusion approximation ($\approx$ SP1);
  - COCAGNE (SPN).
- Solution on more than $10^{11}$ degrees of freedom (DoFs) involved
  - probabilistic solvers (very long computation time);
  - deterministic solvers.

DOMINO (SN) is designed for this validation purpose.
DOMINO: Discrete Ordinates Method In NeutrOnics

- Deterministic, Cartesian, and 3D solver;
- 3 levels of discretization:
  - energy ($G$): multigroup formalism;
  - angle ($\hat{\Omega}$): Level Symmetric Quadrature, $N(N + 2)$ directions
  - space ($x, y, z$): Diamond Differencing scheme (order 0);
- 3 nested levels of iterations:
  - power iterations + Chebychev acceleration;
  - multigroup iterations: Gauss–Seidel algorithm;
  - scattering iterations + DSA acceleration (using the SPN solver):
    → spatial sweep, which consumes most of the computation time.
The Sweep Algorithm

\[
\text{forall the } o \in \text{Octants do}
\]
\[
\text{forall the } c \in \text{Cells do}
\]
\[
\text{forall the } d \in \text{Directions}[o] \text{ do}
\]
\[
\bigtriangledown c = (i,j,k)
\]
\[
\bigtriangledown d = (\nu, \mu, \xi)
\]
\[
\epsilon_x = \frac{2\nu}{\Delta x}; \quad \epsilon_y = \frac{2\eta}{\Delta y}; \quad \epsilon_z = \frac{2\xi}{\Delta z};
\]
\[
\psi[o][c][d] = \frac{\epsilon_x \psi_L + \epsilon_y \psi_B + \epsilon_z \psi_F + S}{\epsilon_x + \epsilon_y + \epsilon_z + \Sigma t};
\]
\[
\psi_R[o][c][d] = 2\psi[o][c][d] - \psi_L[o][c][d];
\]
\[
\psi_T[o][c][d] = 2\psi[o][c][d] - \psi_B[o][c][d];
\]
\[
\psi_{BF}[o][c][d] = 2\psi[o][c][d] - \psi_F[o][c][d];
\]
\[
\phi[k][j][i] = \phi[k][j][i] + \psi[o][c][d] \ast \omega[d];
\]

- 9 add or sub;
- 11 mul;
- 1 div (5 flops)
→ 25 flops per cell, per direction, per energy group.
The Spatial Sweep (*Diamond Differencing scheme*) (1/2)

3D regular mesh with per cell, per angle, per energy group:

- 1 moment to update
- 3 incoming fluxes
- 3 outgoing fluxes
At the beginning, data are known only on the incoming faces
The Spatial Sweep (*Diamond Differencing scheme*) (2/2)

2D example of the spatial mesh for one octant

- **processed cell**
- **ready cell**
The Spatial Sweep *(Diamond Differencing scheme)* (2/2)

2D example of the spatial mesh for one octant

... after a few steps
Parallelization Strategies
Many opportunities for parallelism

- Each level of discretization is a potentially independent computation:
  - energy group
  - angles
  - space
- All energy groups are computed together
- All angles are considered independent
  → This is not true when problems have boundary conditions
- All cell updates on a front are independent
Angular Parallelization Level (Very Low Level)

Several directions belong to the same octant:

- Vectorization of the computation
- Use of SIMD units at processor/core level
  → improve kernel performance
Spatial Parallelization
First level: granularity

Grouping cells in MacroCells:
- Reduces thread scheduling overhead
- Similar to exploiting BLAS 3
- Reduces overall parallelism
Spatial Parallelization

First level: granularity

Grouping cells in **MacroCells**:  
- Reduces thread scheduling overhead  
- Similar to exploiting BLAS 3  
- Reduces overall parallelism
Octant Parallelization
Case of Vacuum Boundary Conditions

When using vacuum boundary conditions, all octants are independent from each other.
Octant Parallelization
Case of Vacuum Boundary Conditions

When using vacuum boundary conditions, all octants are independent from each other.
Octant Parallelization
Case of Vacuum Boundary Conditions

When using vacuum boundary conditions, all octants are independent from each other.
Octant Parallelization
Case of Vacuum Boundary Conditions

Concurrent access to a cell (or MacroCell) are protected by mutexes.
Octant Parallelization
Case of Vacuum Boundary Conditions

Concurrent access to a cell (or MacroCell) are protected by mutexes.
3
Sweep Theoretical Model
Basic formulas

We define the efficiency of the sweep algorithm as follow:

\[ \epsilon = \frac{T_{task} N_{tasks}}{(N_{tasks} + N_{idle}) \times (T_{task} + T_{comm})} \]

\[ = \frac{1}{(1 + N_{idle}/N_{tasks}) \times (1 + T_{comm}/T_{task})} \]

Objective: \textbf{Minimize} \quad N_{idle}
For 3D block distribution

The minimal number of idle steps are those required to reach the cube center:

\[ N_{\text{idle}}^{\text{min}} = P_x + \delta_x - 2 + P_y + \delta_y - 2 + P_z + \delta_z - 2 \]

where \( \delta_u = 0 \), if \( P_u \) is even, 1 otherwise.

Objective: **Minimize the sum** \( P + Q + R \), where \( P \times Q \times R \) is the process grid.

\( \rightarrow \) Hybrid MPI-Thread implementation allows this
Hybrid Model
4
DOMINO on top of PARSEC
DOMINO on top of PaRSEC

Implementation

- Only one kind of task:
  - Associated to one MacroCell
  - All energy group
  - All directions included in one octant
    → 8 tasks per MacroCell
  - No dependencies from one octant to another
    → protected by mutexes

- Simple algorithm to write in JDF

- Require a data distribution:
  - Independent from the algorithm: 2D, 3D, cyclic or not, . . .
  - For now: Block-3D (Non cyclic) with a $P \times Q \times R$ grid

- Fluxes on faces are dynamically allocated/freed by the runtime
DOMINO JDF Representation (2D)

```c
1  CellUpdate(a, b)
2
3  /* Execution Space */
4  a = 0 .. ncx-1
5  b = 0 .. ncy-1
6
7  /* Task Locality (Owner Compute) */
8  : mcg(a, b)
9
10  /* Data dependencies */
11  RW X <- (a != aBeg) ? X CellUpdate(a-alnc, b) : X READ_X(b)
12     -> (a != aEnd) ? X CellUpdate(a+aLnc, b)
13  RW Y <- (b != bBeg) ? Y CellUpdate(a, b-bLnc) : Y READ_Y(a)
14     -> (b != bEnd) ? Y CellUpdate(a, b+bLnc)
15  RW MCG <- mcg(a, b)
16      -> mcg(a, b)
17  BODY
18  {
19     solve( MCG, X, Y, ... );
20  }
21  END
```

- `aBeg, aEnd, alnc, bBeg, bEnd` and `blnc` are octant dependent variables.
5

Results
Scalability of the existing implementation with Intel TBB
32-core Nehalem node with two 4-way SIMD units running at 2.26 Ghz

- 2 energy groups calculation;
- S8 Level Symmetric quadrature (80 angular directions);
- spatial mesh: $120 \times 120 \times 120$ and $480 \times 480 \times 480$. 

![Graph showing scalability results](image-url)
DOMINO on top of PaRSEC

Shared Memory Results: Comparison with Intel TBB

- 1 energy group;
- mesh size: $480 \times 480 \times 480$;
- Level Symmetric S2;
- 7.9 Gflops (4.6%)
DOMINO on top of PaRSEC

Shared Memory Results: Comparison with Intel TBB

- 1 energy group;
- mesh size: 480 × 480 × 480;
- Level Symmetric S8;
- 57.2 Gflops (33.5%)
DOMINO on top of PaRSEC

Shared Memory Results: Comparison with Intel TBB

- 1 energy group;
- mesh size: $480 \times 480 \times 480$;
- Level Symmetric S16;
- 92.6 Gflops (54.2%)
DOMINO on top of PaRSEC

Shared Memory Results: Comparison with Intel TBB

Test on manumanu NUMA node: 160 cores.
DOMINO on top of PaRSEC
Distributed Memory Results (Ivanoe)

- 1 energy group; mesh size: $480 \times 480 \times 480$; Level Symmetric S16;
DOMINO on top of PaRSEC
Distributed Memory Results (Athos)

- 1 energy group; mesh size: $480 \times 480 \times 480$; Level Symmetric S16;
Results

DOMINO on top of PaRSEC
Distributed Memory Results (Athos)

- 1 energy group; mesh size: $120 \times 120 \times 120$; *Level Symmetric* S16;
DOMINO on top of PaRSEC
Distributed Memory Results

Execution trace for a run on 8 nodes (2, 2, 2) (Bad scheduling).
DOMINO on top of PaRSEC
Distributed Memory Results

Execution trace for a run on 8 nodes (2, 2, 2) (Good scheduling).
DOMINO on top of PaRSEC
Scheduling by front

Disco  NoPrio  Prio
6
Conclusion and future works
Conclusion and Future Work

Conclusion

- Efficient implementation on top of PaRSEC
  - Less than 2 weeks to be implemented
  - Comparable to Intel TBB in shared memory
- Multi-level implementation:
  - Code vectorization (angular direction)
  - Block algorithm (MacroCells)
  - Hybrid MPI-Thread implementation

Future work

- Finish the hybrid model to get better evaluation of the performance
- Experiments on Intel Xeon Phi
Thanks !