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

S. Moustafa, M. Faverge
L. Plagne, and P. Ramet
LabRI – Inria Bordeaux Sud-Ouest
R&D – SINETICS

July 1, 2014

1
Context and goals

Guideline

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

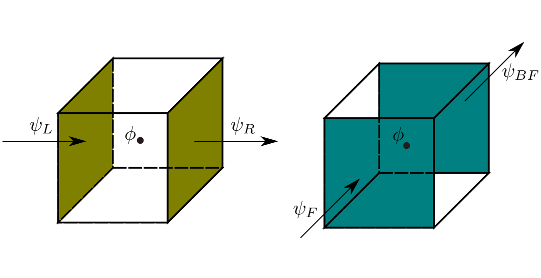
- Deterministic, Cartesian, and 3D solver;
- 3 levels of discretization:
  - energy (G): multigroup formalism;
  - angle ($\vec{\Omega}$): *Level Symmetric Quadrature*, $N(N+2)$ directions
  - space (x, y, z): *Diamond Differencing scheme* (order 0);
- 3 nested levels of iterations:
  - power iterations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

The Sweep Algorithm

```
forall the o ∈ Octants do
    forall the c ∈ Cells do
        ▷ c = (i, j, k)
        forall the d ∈ Directions[o] do
            ▷ d = (ν, μ, ξ)
            ε_x = 2ν/Δx;   ε_y = 2η/Δy;   ε_z = 2ξ/Δz;
            ψ[o][c][d] = (ε_x ψ_L + ε_y ψ_B + ε_z ψ_F + S) / (ε_x + ε_y + ε_z + Σ_t);
            ψ_R[o][c][d] = 2ψ[o][c][d] − ψ_L[o][c][d];
            ψ_T[o][c][d] = 2ψ[o][c][d] − ψ_B[o][c][d];
            ψ_BF[o][c][d] = 2ψ[o][c][d] − ψ_F[o][c][d];
            φ[k][j][i] = φ[k][j][i] + ψ[o][c][d] ∗ ω[d];
        end
    end
end
```

- 9 add or sub;
- 11 mul;
- 1 div (5 flops)
  $\rightarrow$ 25 flops per cell, per direction, per energy group.

The Spatial Sweep (*Diamond Differencing scheme*) (1/2)



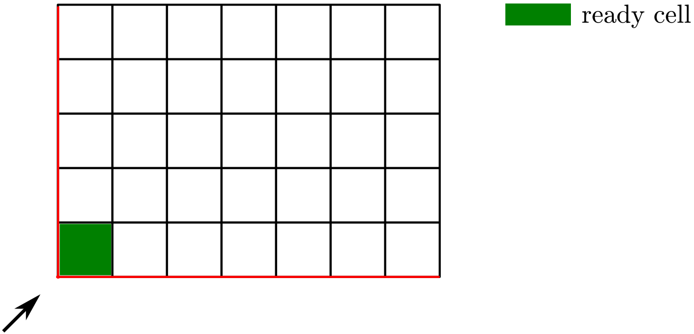
3D regular mesh with per cell, per angle, per energy group:

- 1 moment to update
- 3 incoming fluxes
- 3 outgoing fluxes

The Spatial Sweep (*Diamond Differencing scheme*) (2/2)

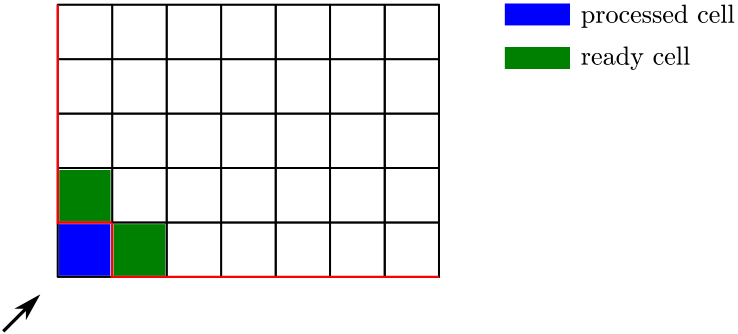
2D example of the spatial mesh for one octant

At the beginning, data are known only on the incoming faces



The Spatial Sweep (*Diamond Differencing scheme*) (2/2)

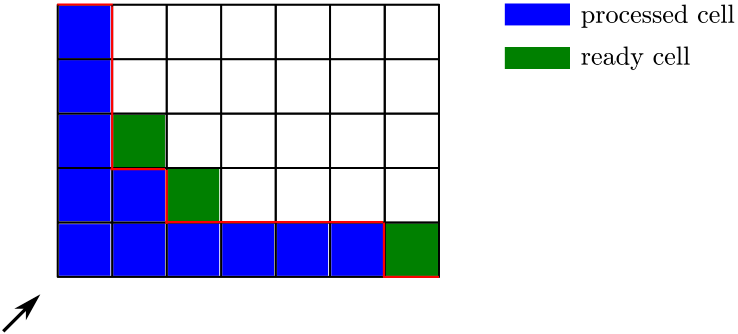
2D example of the spatial mesh for one octant



The Spatial Sweep (*Diamond Differencing scheme*) (2/2)

2D example of the spatial mesh for one octant

... after a few steps

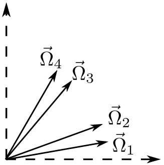


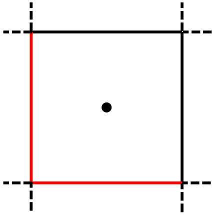
2
Parallelization Strategies

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
  $\rightarrow$ This is not true when problems have boundary conditions
- All cell updates on a front are independent

Angular Parallelization Level (Very Low Level)



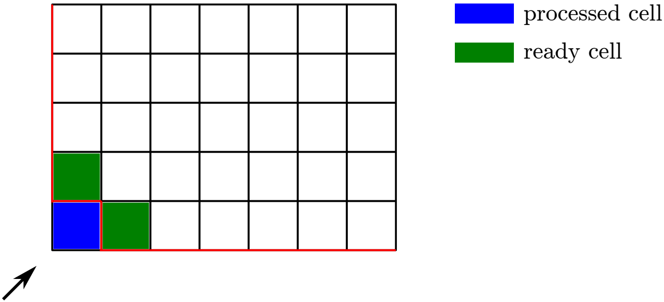


Several directions belong to the same octant:

- Vectorization of the computation
- Use of SIMD units at processor/core level
  $\rightarrow$ improve kernel performance

Spatial Parallelization

First level: granularity

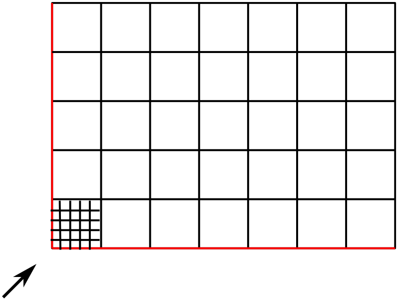


Grouping cells in MacroCells:

- Reduces thread scheduling overhead
- Similar to exploiting BLAS 3
- Reduces overall parallelism

Spatial Parallelization

First level: granularity



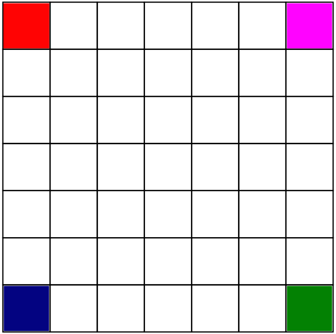
Grouping cells in MacroCells:

- Reduces thread scheduling overhead
- Similar to exploiting BLAS 3
- Reduces overall parallelism

Octant Parallelization

Case of Vacuum Boundary Conditions

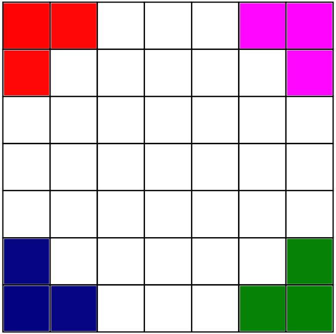
When using vacuum boundary conditions, all octants are independent from each other



Octant Parallelization

Case of Vacuum Boundary Conditions

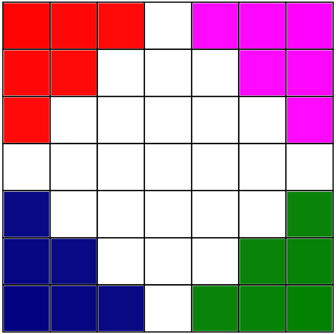
When using vacuum boundary conditions, all octants are independent from each other



Octant Parallelization

Case of Vacuum Boundary Conditions

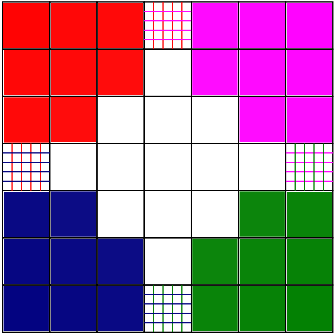
When using vacuum boundary conditions, all octants are independent from each other



Octant Parallelization

Case of Vacuum Boundary Conditions

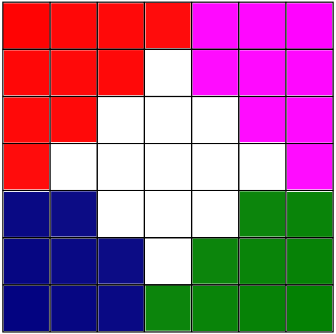
Concurrent access to a cell (or MacroCell) are protected by mutexes.



Octant Parallelization

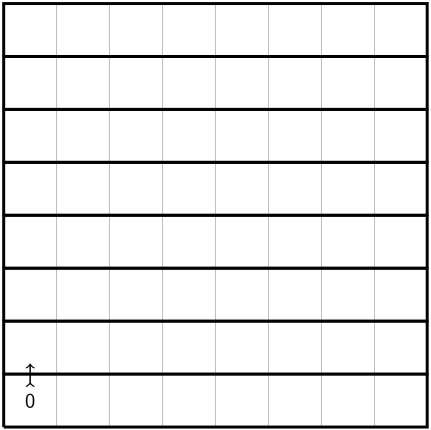
Case of Vacuum Boundary Conditions

Concurrent access to a cell (or MacroCell) are protected by mutexes.

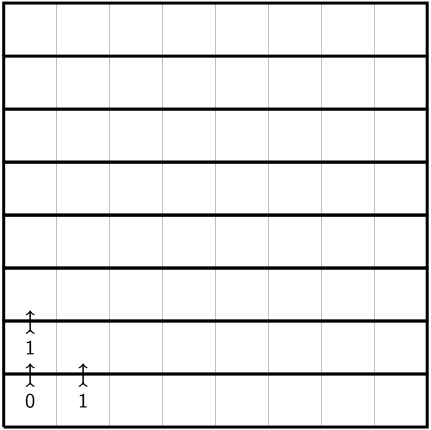


3
Sweep Theoritical Model

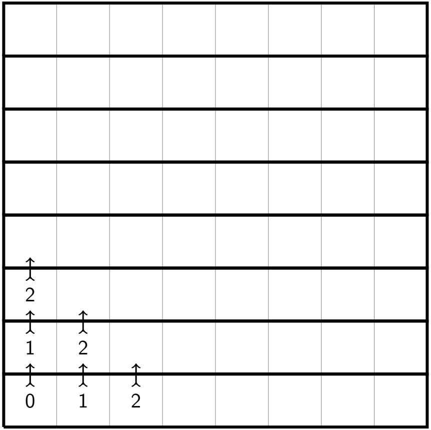
Basic ideas - Flat Model



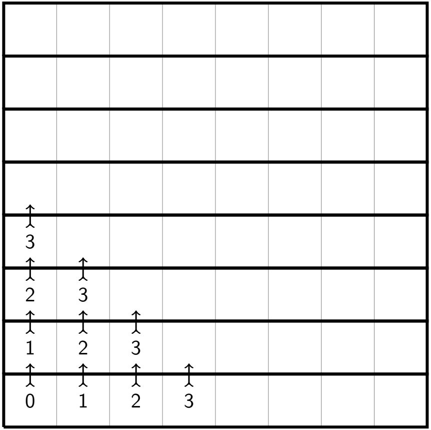
Basic ideas - Flat Model



Basic ideas - Flat Model



Basic ideas - Flat Model



- 1D block distribution
- Requires:
  - 14 tasks
  - 7 communications

Basic ideas - Flat Model

7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11
3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7

- 1D block distribution
- Requires:
  - 14 tasks
  - 7 communications

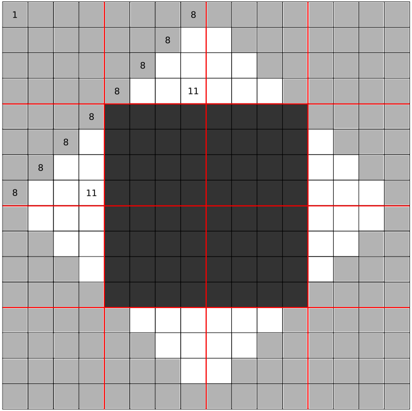
Formulas (Adams et al.)

We define the efficiency of the sweep algorithm as follow:

$$\begin{aligned}\epsilon &= \frac{T_{task} N_{tasks}}{(N_{tasks} + N_{idle}) * (T_{task} + T_{comm})} \\ &= \frac{1}{(1 + N_{idle}/N_{tasks}) * (1 + T_{comm}/T_{task})}\end{aligned}$$

Objective: **Minimize** N_{idle}

Filling the pipeline



For 3D block distribution

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

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

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 should allow this

Hybrid MPI-Thread model

7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11
3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7

- Requires:
  - 14 tasks
  - 2 communications instead of 7

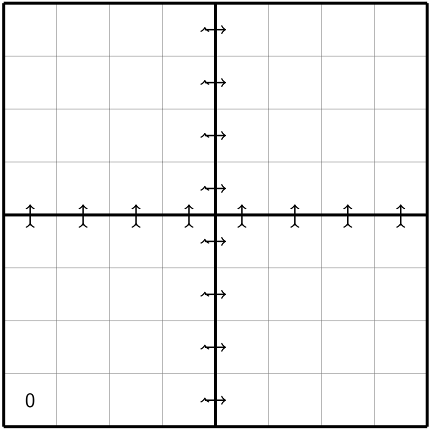
Hybrid MPI-Thread model

7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11
3	4	5	6	7	8	9	10
2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7

- Requires:
  - 14 tasks
  - 2 communications instead of 7
- Only 2 cores per node!!!

Hybrid MPI-Thread model

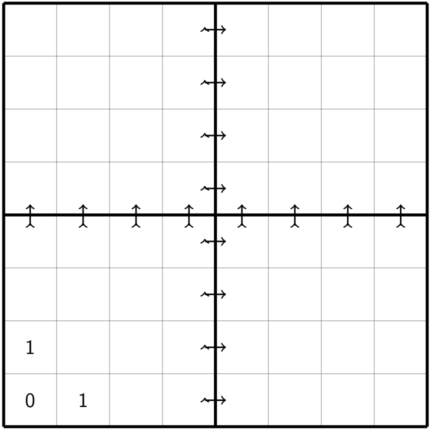
Scheduling by front



- Natural order: follow the fronts
- Requires 19 steps

Hybrid MPI-Thread model

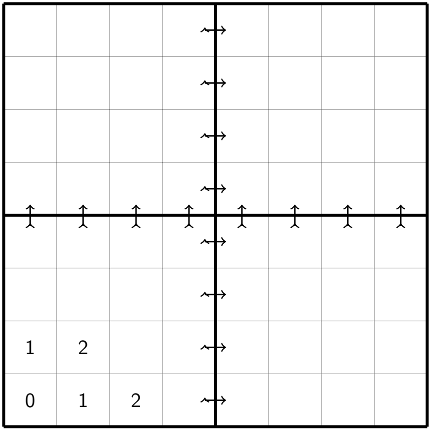
Scheduling by front



- Natural order: follow the fronts
- Requires 19 steps

Hybrid MPI-Thread model

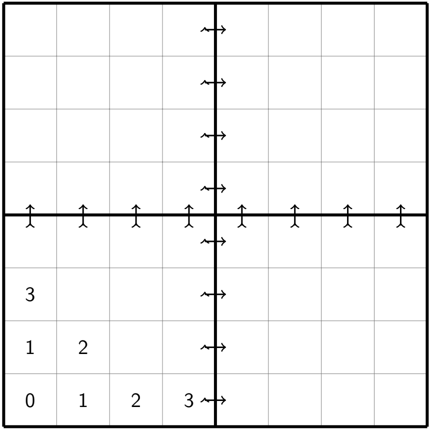
Scheduling by front



- Natural order: follow the fronts
- Requires 19 steps

Hybrid MPI-Thread model

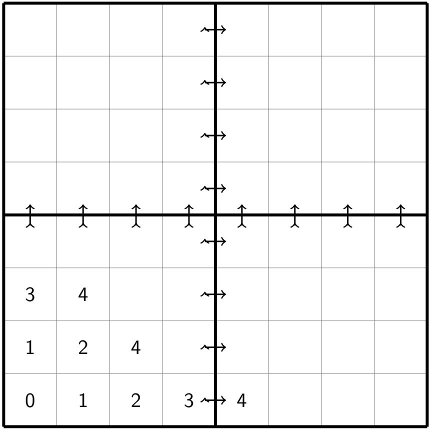
Scheduling by front



- Natural order: follow the fronts
- Requires 19 steps

Hybrid MPI-Thread model

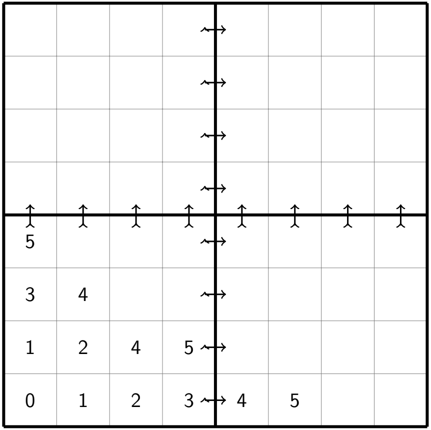
Scheduling by front



- Natural order: follow the fronts
- Requires 19 steps

Hybrid MPI-Thread model

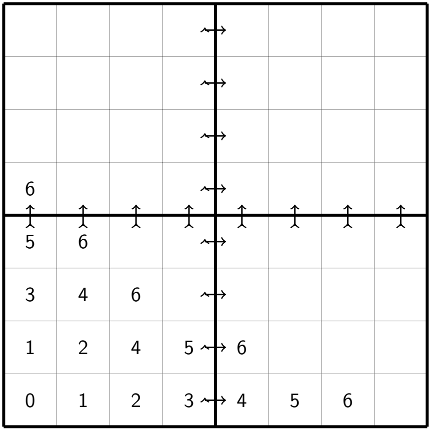
Scheduling by front



- Natural order: follow the fronts
- Requires 19 steps

Hybrid MPI-Thread model

Scheduling by front



- Natural order: follow the fronts
- Requires 19 steps

Hybrid MPI-Thread model

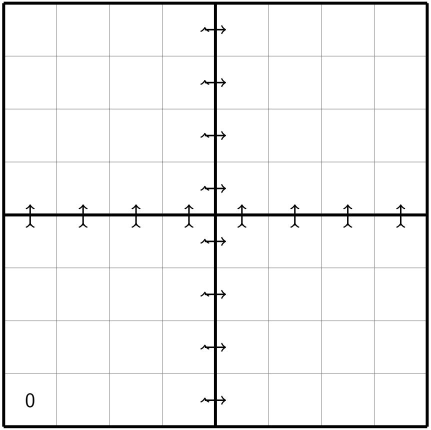
Scheduling by front

11	12	13	14	16	17	18	19
9	10	12	13	14	15	17	18
7	8	10	11	12	13	15	16
6	7	8	9	11	12	13	14
5	6	7	8	10	11	12	13
3	4	6	7	8	9	11	12
1	2	4	5	6	7	9	10
0	1	2	3	4	5	6	8

- Natural order: follow the fronts
- Requires 19 steps

Hybrid MPI-Thread model

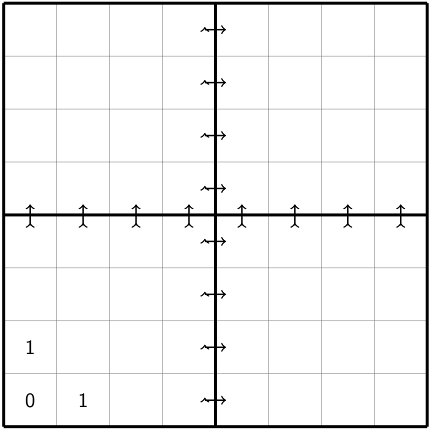
Favour one direction



- Give priority to one direction of the octant
- Might delay other directions
- Requires **18** steps

Hybrid MPI-Thread model

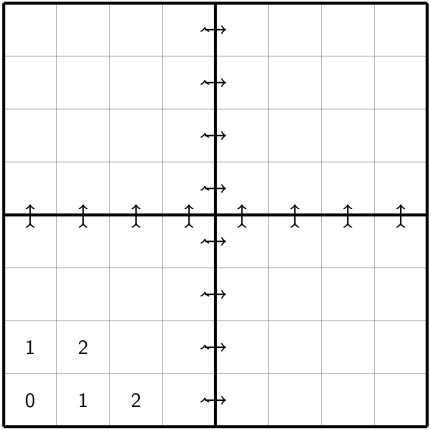
Favour one direction



- Give priority to one direction of the octant
- Might delay other directions
- Requires **18** steps

Hybrid MPI-Thread model

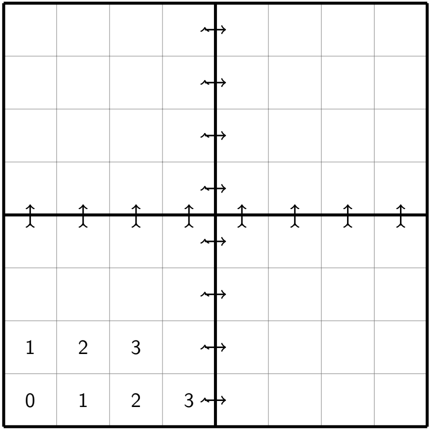
Favour one direction



- Give priority to one direction of the octant
- Might delay other directions
- Requires **18** steps

Hybrid MPI-Thread model

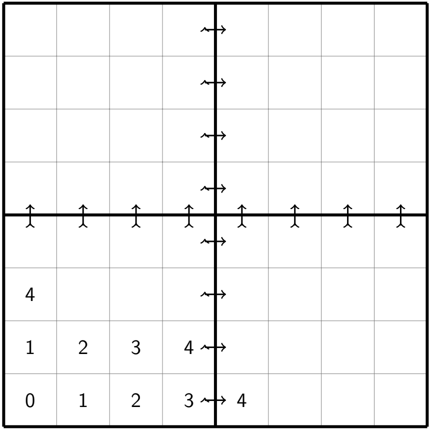
Favour one direction



- Give priority to one direction of the octant
- Might delay other directions
- Requires **18** steps

Hybrid MPI-Thread model

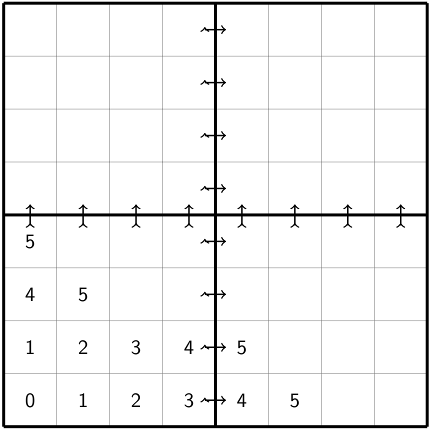
Favour one direction



- Give priority to one direction of the octant
- Might delay other directions
- Requires **18** steps

Hybrid MPI-Thread model

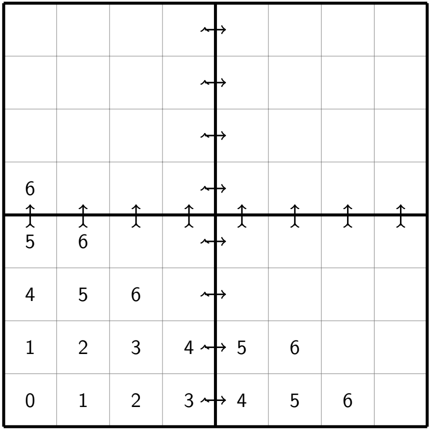
Favour one direction



- Give priority to one direction of the octant
- Might delay other directions
- Requires **18** steps

Hybrid MPI-Thread model

Favour one direction



- Give priority to one direction of the octant
- Might delay other directions
- Requires **18** steps

Hybrid MPI-Thread model

Favour one direction

11	12	13	14	15	16	17	18
10	11	12	13	14	15	16	17
7	8	9	10	11	12	13	14
6	7	8	9	10	11	12	13
5	6	7	8	9	10	11	12
4	5	6	7	8	9	10	11
1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7

- Give priority to one direction of the octant
- Might delay other directions
- Requires **18** steps

Hybrid MPI-Thread model

Priority used

For 3D distribution grid $P \times Q \times R$ with $P > Q > R$, we favour the largest direction first.

4
DOMINO on top of PaRSEC

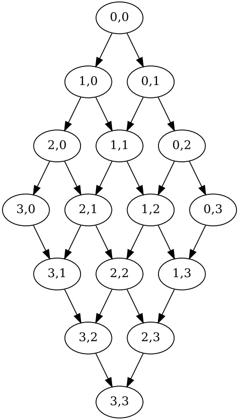
DOMINO on top of PaRSEC

Implementation

- Only one kind of task:
  - Associated to one MacroCell
  - All energy group
  - All directions included in one octant
    $\rightarrow$ 8 tasks per MacroCell
  - No dependencies from one octant to another
    $\rightarrow$ 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 (1 sweep in 2D)

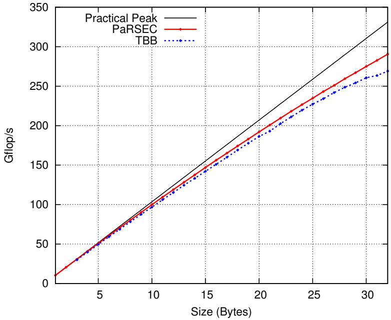
```
T(a, b)
// Execution space
a = 0 .. 3
b = 0 .. 3

// Parallel partitioning
: mcg(a, b)

// Parameters
RW X   <- (a != 0) ? X  T(a-1, b)
       -> (a != 3) ? X  T(a+1, b)

RW Y   <- (b != 0) ? Y  T(b, b-1)
       -> (b != 3) ? Y  T(b, b+1)

RW MCG <- mcg(a, b)
       -> mcg(a, b)

BODY
{
  computePhi ( MCG, X, Y, ... );
}
END
```



5
Results

Shared Memory Results (PaRSEC VS Intel TBB)

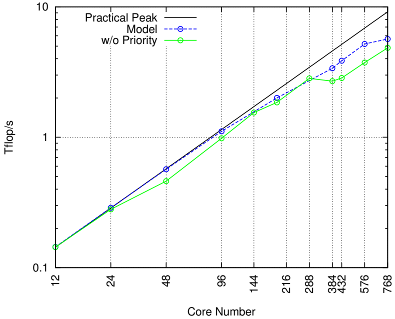
32 cores – Intel X7560



- Mesh size: $480 \times 480 \times 480$; *Level Symmetric* S16 (288 directions)
- Achieves 291 Gflop/s (51% of Theoretical Peak Perf.)

Distributed Memory Results – Hybrid

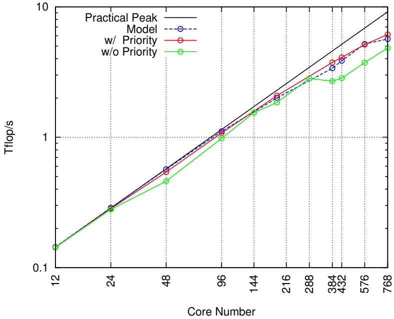
IVANOE – 768 cores (64 nodes of 12 cores) – Intel X7560



- Mesh size: $480 \times 480 \times 480$; *Level Symmetric* S16 (288 directions)
- Parallel efficiency: 52.7%
- 4.8 Tflop/s (26.8% of Theoretical Peak Perf.) at 768 cores

Distributed Memory Results – Hybrid

IVANOE – 768 cores (64 nodes of 12 cores) – Intel X7560

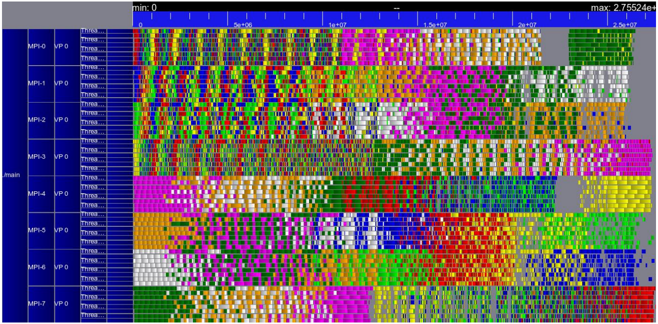


- Mesh size: $480 \times 480 \times 480$; *Level Symmetric* S16 (288 directions)
- Parallel efficiency: 66.8%
- 6.2 Tflop/s (34.4% of Theoretical Peak Perf.) at 768 cores

Distributed Memory Results – Hybrid

Execution traces

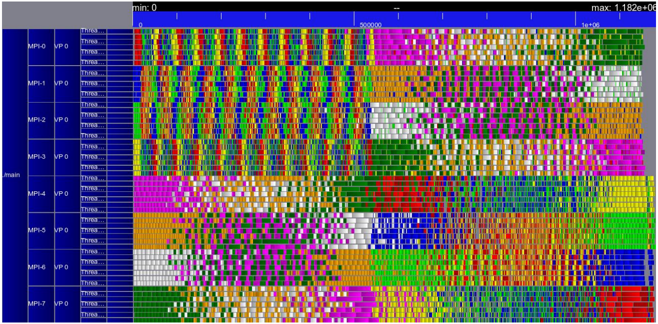
Execution trace for a run on 8 nodes (2, 2, 2) (w/o priorities).



Distributed Memory Results – Hybrid

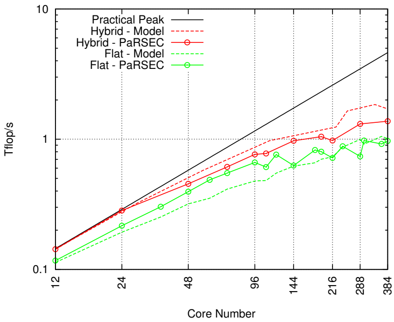
Execution traces

Execution trace for a run on 8 nodes (2, 2, 2) (w/ priorities).



Distributed Memory Results – Flat vs Hybrid

IVANOE – 384 cores – Intel X7560



- Mesh size: $120 \times 120 \times 120$; *Level Symmetric* S16 (288 directions)
- Flat model: Overlap is not integrated into the model

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
- *Simple* multi-level implementation:
  - Code vectorization (angular direction)
  - Block algorithm (MacroCells)
  - Hybrid MPI-Thread implementation

Future work

- Fix the hybrid model to try new scheduling and get the best data distribution out of it
- Experiments on Intel Xeon Phi
- Model of the symmetric case

Thanks !