

Nested parallelism applied to AMRCLAW

Ragnhild Blikberg
Parallab/ Department of Informatics
University of Bergen
NORWAY

Outline

- Adaptive Mesh Refinement (AMR)
- AMRCLAW
- Testproblem: Shallow water equations
- 1-level parallel approaches
- Nested parallel approach
- Numerical results
- Implementation
- Conclusions

Two ways of improving performance used:

1. Adaptive Mesh Refinement (AMR)

- Mesh refinement
- Adaptivity

2. Parallelization

- 1-level fine-grained
- 1-level coarse-grained
- Nested

AMR

When solving PDE numerically, the need for dense grid points may differ from one subdomain to another.

In AMR grid points are clustered adaptively in regions where it is most needed of close grid points.

One can have a fine resolution where it is needed and use a coarse resolution where this is sufficient.

- A level L is a union of grid *patches* of a specific resolution.
- NL : user defined max level of refinement
- All grids regenerated every K time step.
- Cells with error $> tol$ flagged for refinement.
- R_{L+1} : refinement factor
- Refined patches are integrated R_{L+1} time steps, $dT_{L+1} = dT_L / R_{L+1}$, to catch up.

The recursive AMR algorithm

(1 coarse grid time step, $R_1 = 1$)

AMR(L)

For $T_L = 1, R_L$

If (k th time step) and ($L < NL$) **Regrid**

Compute boundary values

Advance PDE 1 time step

If ($L < NL$) AMR($L+1$)

End for

Coarse grid update

Conservation fixup

The regrid algorithm

Every k th time step for $L < NL$

Compute boundary values

Advance PDE 1 time step

- for existing grid
- for grid twice as coarse

Estimate error

Flag cells with error $> tol$

Organise flagged cells into patches

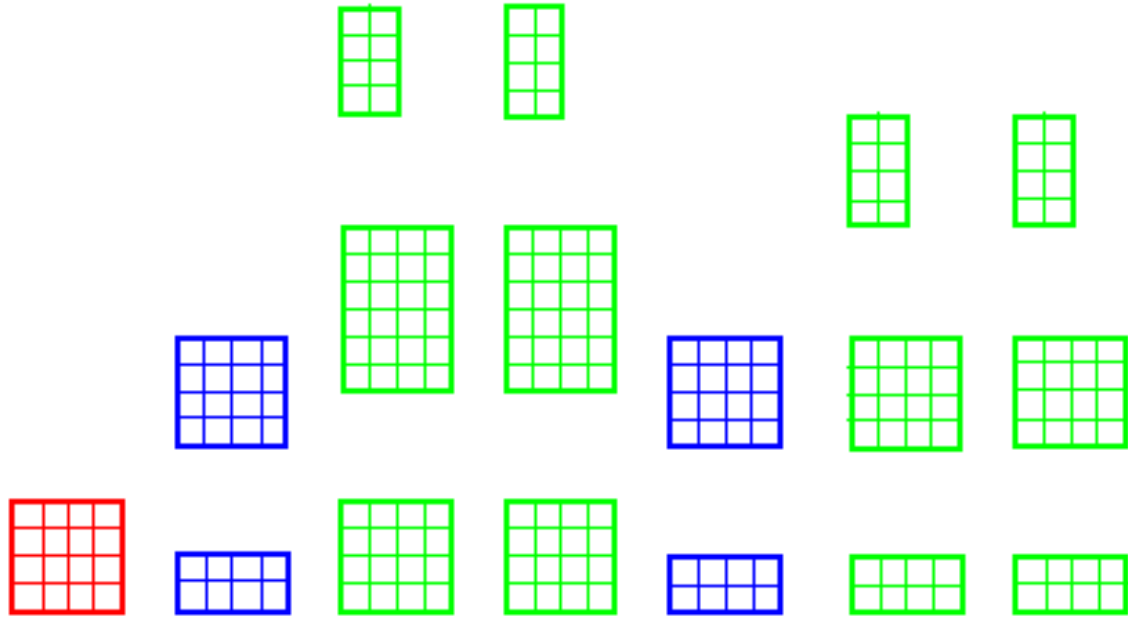
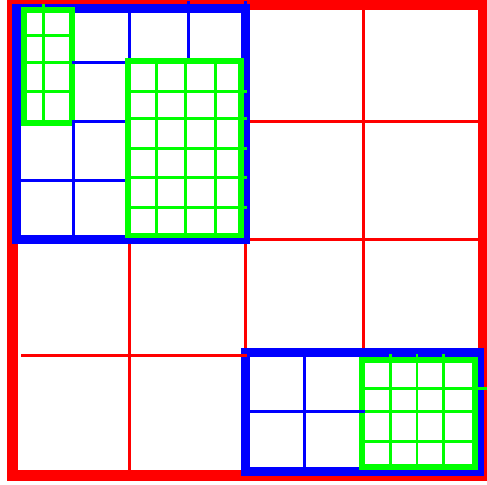
Refine patches with factor R_{L+1}

Example

$$NL = 3$$

$$R_2 = R_3 = 2$$

$$K = 1$$



AMRCLAW

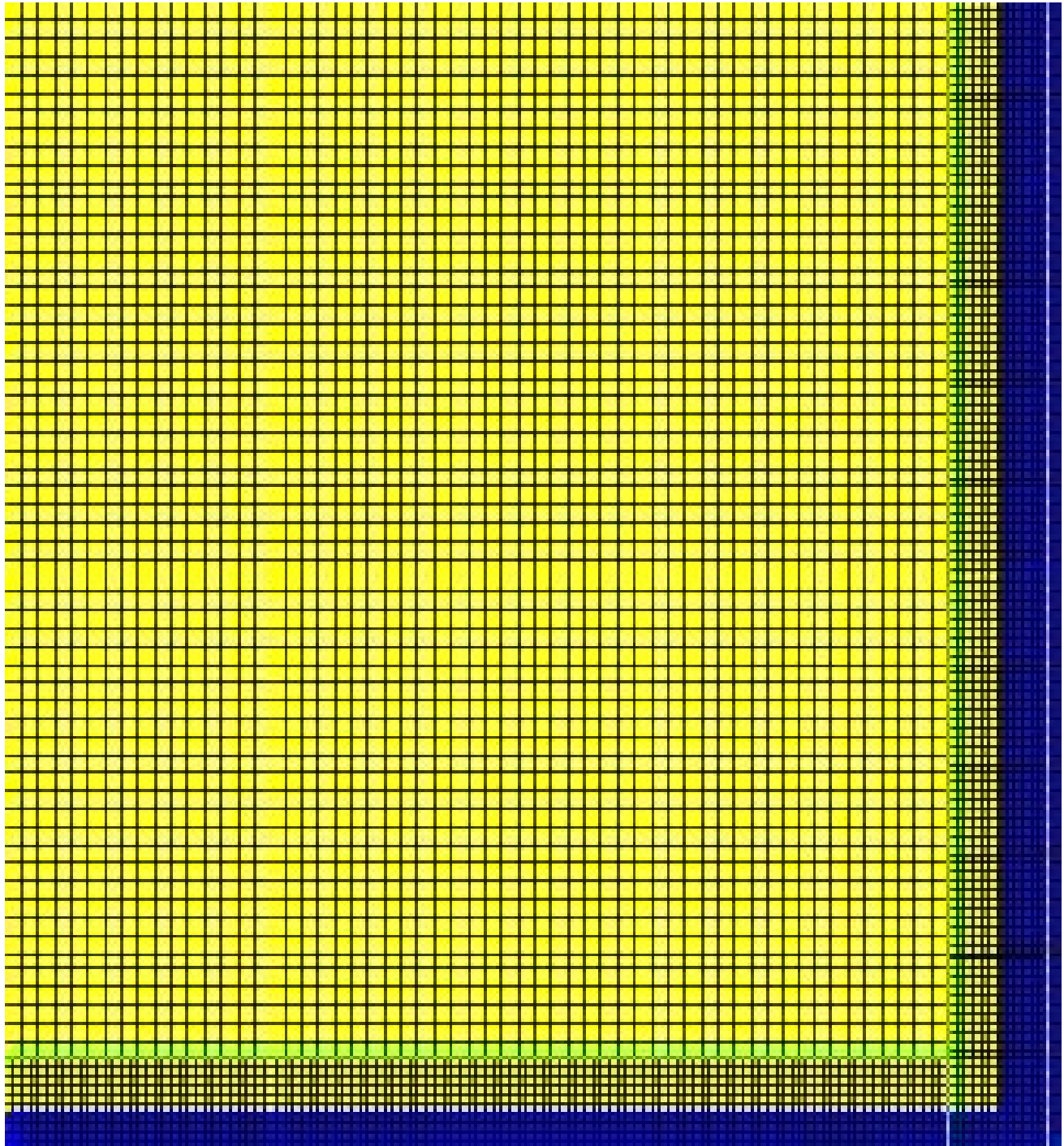
- freely available software package in Fortran 77 created by Randy LeVeque and Marsha Berger.
- based on Bergers AMR algorithms and LeVeques software Conservation LAW PACKage, CLAWPACK.
- Now a part of CLAWPACK 4.0.

What the user has to specify in AMRCLAW?

- input parameters
- initial conditions
- boundary conditions
- Riemann-solver
- source term
- ...

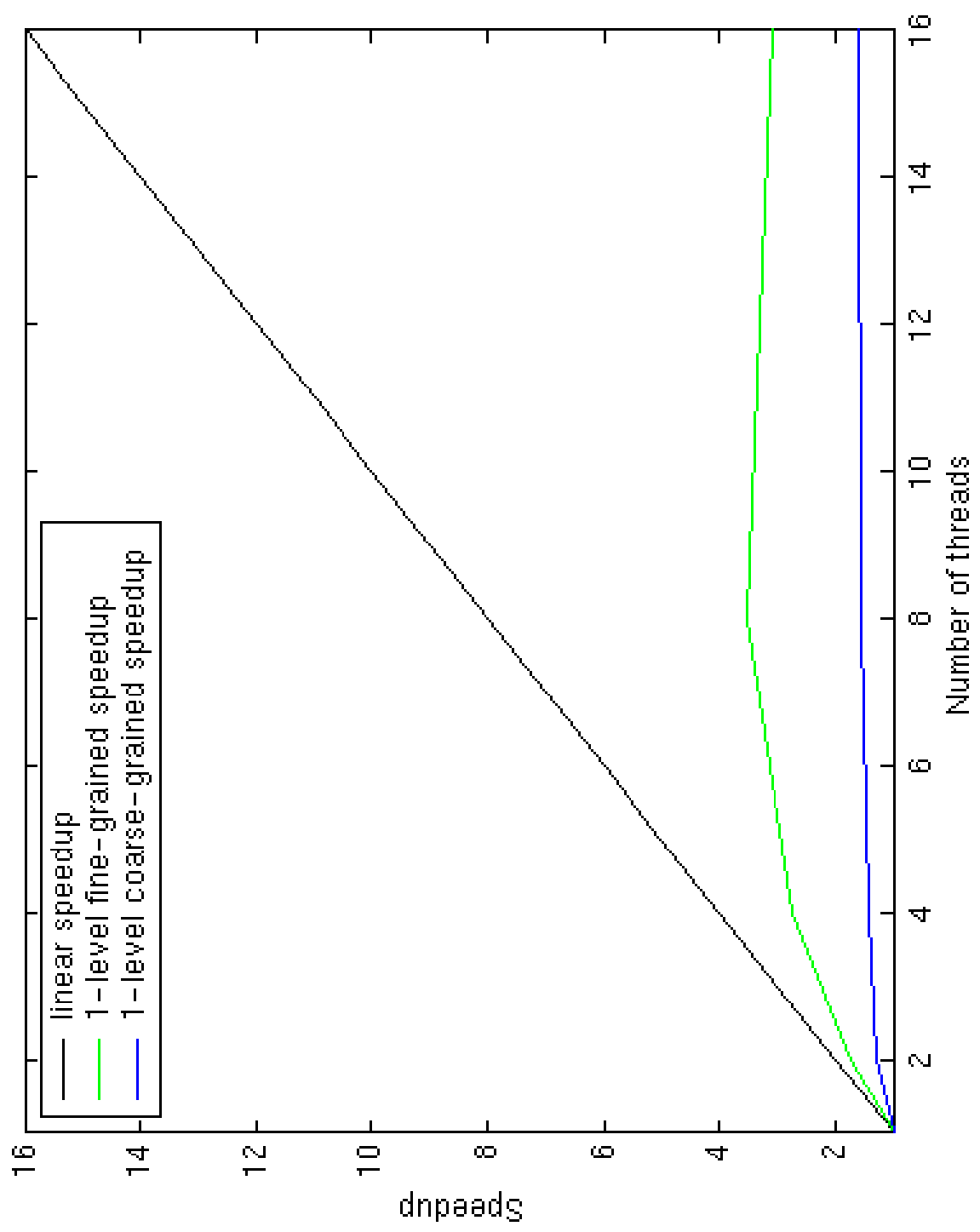
Test problem: Shallow water equations

- 20km x 20km basin filled with water
- Solid walls with no friction
- Flat bottom topography
- Equilibrium depth = 200m
- Coarse grid: 128 x 128
- $NL = 2$, $tol = 0.06$, $R_2 = 2$, $K = 1$



Parallel approaches

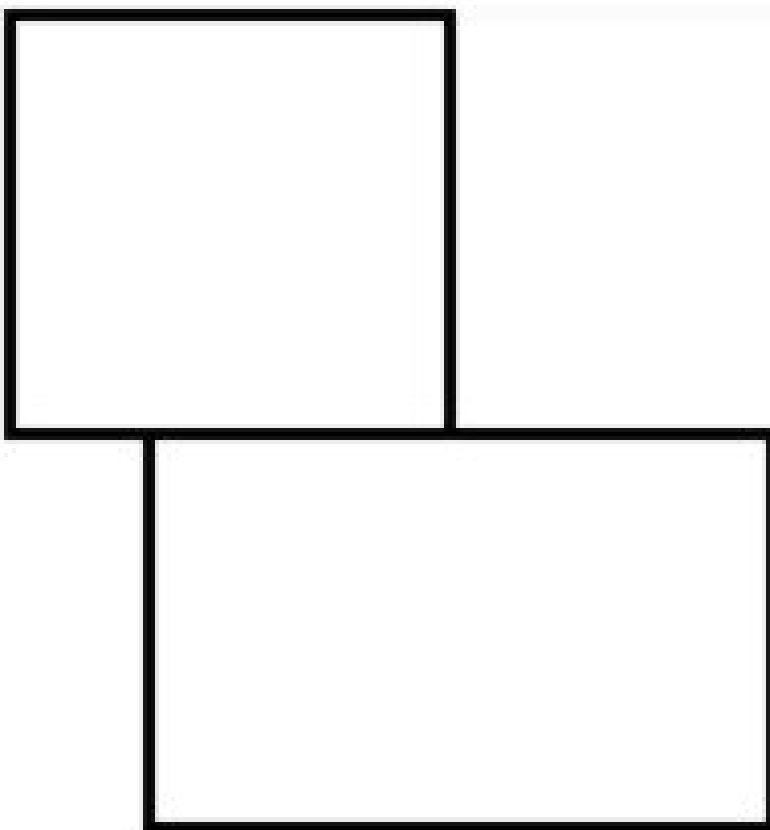
- 1-level parallelization of AMRCLAW
 - Fine-grained on loop-level
 - All threads work on the same patch(es)
 - Coarse-grained on patch-level
 - Each patch only executed by one thread



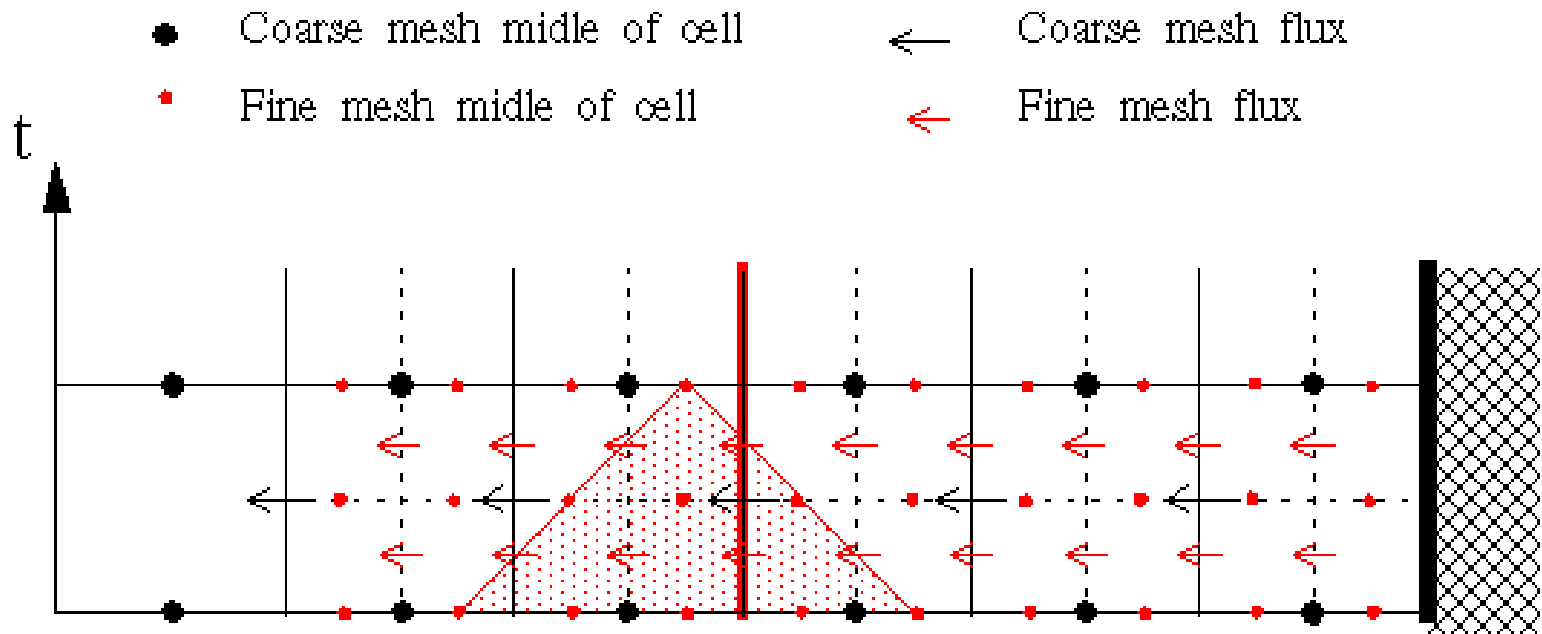
Removing dependency

To avoid synchronization between patches at level NL , it was necessary to remove a dependency in the assigning of boundary values.

Originally, values are copied from neighbor patches if these exists. If not, they are linear interpolated from coarser grid.



Alternative: integrate 1 extra cell in each direction in the first out of R_{NL} time steps.



Nested parallelism applied to AMRCLAW

The fine-grained and coarse-grained approaches complete each other.

We wanted to make a nested approach mixing the 1-level approaches by utilizing the best qualities of both in a flexible manner.

Distribution algorithm

- distributes the work, $w_{1:N}$, of N tasks (patches) to P threads.
- divides the patches in 2 sets, L and S .
 - L : one or more threads are working on the same task
 - S : each thread has one or more tasks to work on

$$[p_{1:T}] = \text{Distribute}(N, P, w_{1:N})$$

- T = number of teams
- p_i = number of threads in team i

Nested parallel AMR algorithm

(Time consuming part)

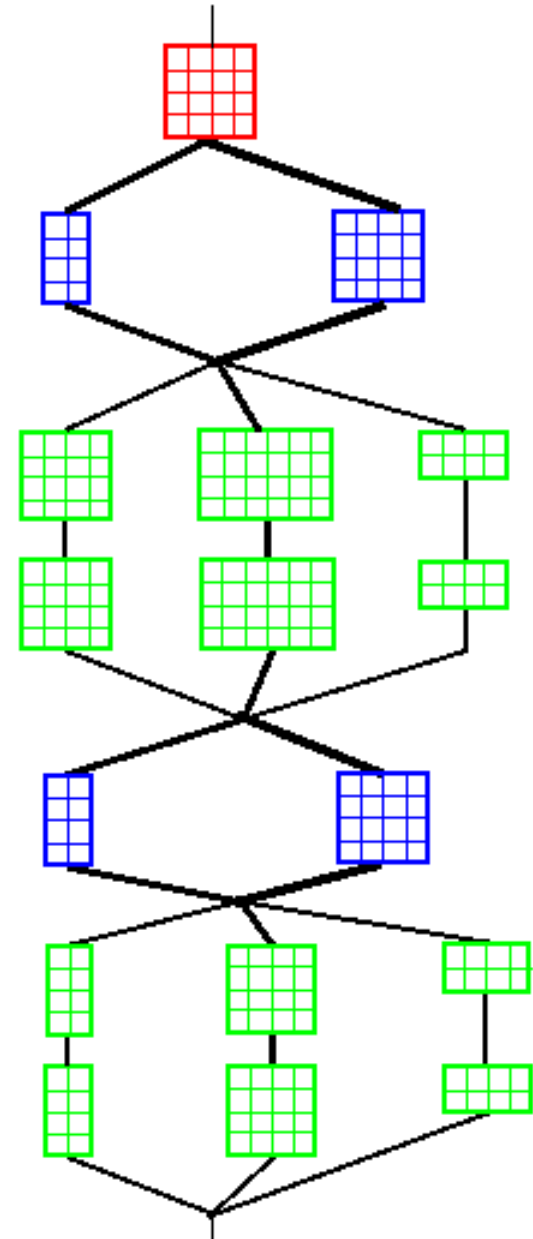
```

    [p1:T] = Distribute(N, P, w1:N)
!$rb
Parallel Teams(p1:T)
!$omp
Parallel
    Compute boundary values
    Advance PDE 1 time step
!$omp
End Parallel
!$rb
End Parallel Teams
```

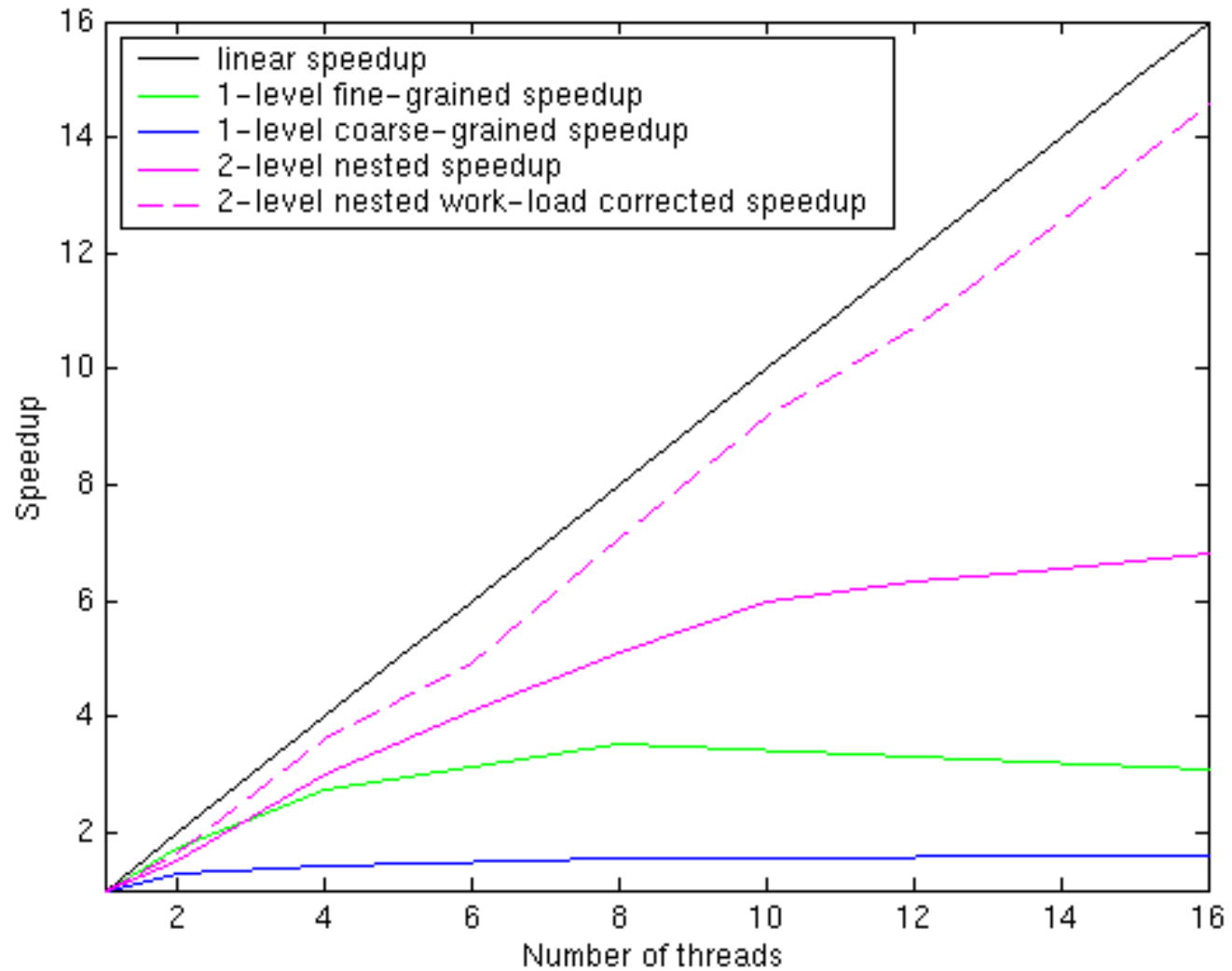
Parallel example

3 levels of refinement

2 levels of parallelization



Numerical results ($NL=2$)



Numerical results

- Nested parallel version has much better speedup than 1-level parallel versions.
- "Nested overhead" neglectable
- Dynamic structure implies non-uniform data access, which probably is the reason for the moderate speedup.

Implementation

- In lack of nesting implemented in OpenMP-compiler, explicit thread programming used
- Missed **!\$OMP TEAMPRIVATE**
- Missed **!\$OMP TEAMBARRIER**
- Missed "partial synchronization" directive
- Extra changes in the code was necessary in lack of these directives.

Conclusions

- ◆ More difficult to obtain scalability on problems with dynamic changing number of tasks.
- ◆ Nevertheless, nested parallelism much better than 1-level parallelism.