Nested Parallelism in OpenMP

Ragnhild Blikberg
Para//ab
University of Bergen
NORWAY

Joint work with Prof. Tor Sørevik

Overview

- Introduction
- Experiments - presentation
- Load balancing
- Implementation
- Experiments - results
- OpenMP
- Conclusions

Introduction

Q: When is nested parallelism desirable?

A: For computational problems having

- an outer level of coarse-grained parallelism, where the number of tasks is relatively small, and
- where each task itself has an inner level of fine-grained parallelism.

Experiments - presentation

We will test the effect of nested parallelism on two real life problems:

- A wavelet based data compression routine
  - having a static number of tasks
- An adaptive mesh refinement (AMR) application
  - having a dynamically changing number of tasks

A wavelet based data compression routine

Data is transformed into wavelet-space using a 2d-wavelet transform, where only non-zero wavelet coefficients are stored.

A 2d-wavelet transform is done by applying multiple, independent 1d-wavelet transforms to each row in the block matrix, and next to the columns.

The wavelet routine only works for arrays $m \times n$ where m and n are integers power of 2. Thus, the array is chopped up in power-of-2 blocks. For each of these blocks a 2d-wavelet transform is carried out.

Experiments - presentation: Wavelet test case

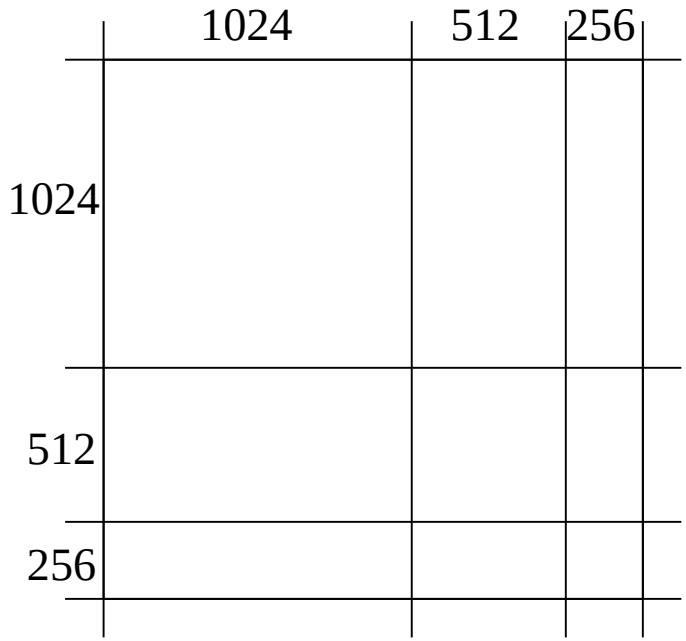


Figure 1. *Wavelet test case: a 2d array of size* 1792×1792.

An adaptive mesh refinement (AMR) application

Finding the numerical solution to real life problems modeled by PDEs, the need for dense grid points may differ from region to region and sometimes also in time.

Using the resolution needed to adequately resolve these features on a uniform grid will in most cases be too expensive. The answer is to refine the grid only where necessary.

If it is not known in advance where high resolution is needed, adaptivity is called for. AMR will dynamically create and remove patches of refined mesh as the need for fine resolution changes.

Experiments - presentation: AMR test case

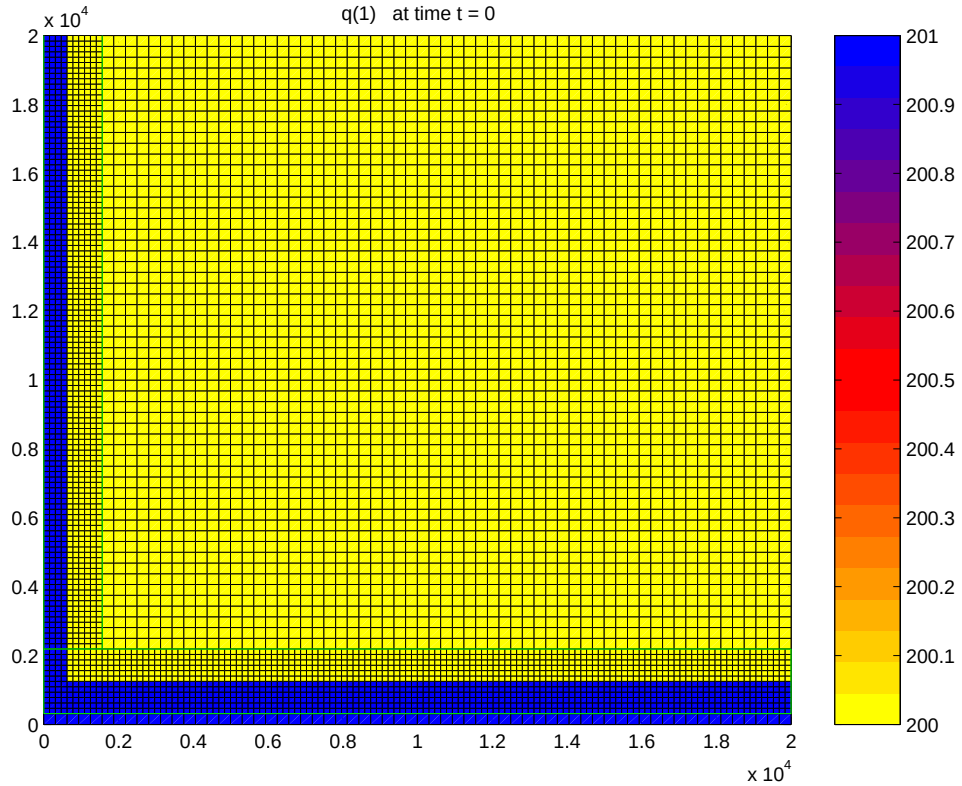
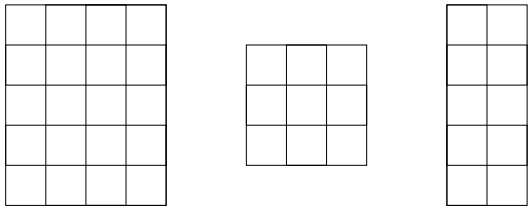


Figure 2. *AMR test case: a dam break problem, using a 2-level grid*

Experiments - presentation: AMR test case

As a motivating example, let us compare

- a 1-level fine-grained parallelization
- a 1-level coarse-grained parallelization
- a 2-level nested parallelization



$W = 20 + 9 + 10 = 39$

Experiments - presentation: AMR test case

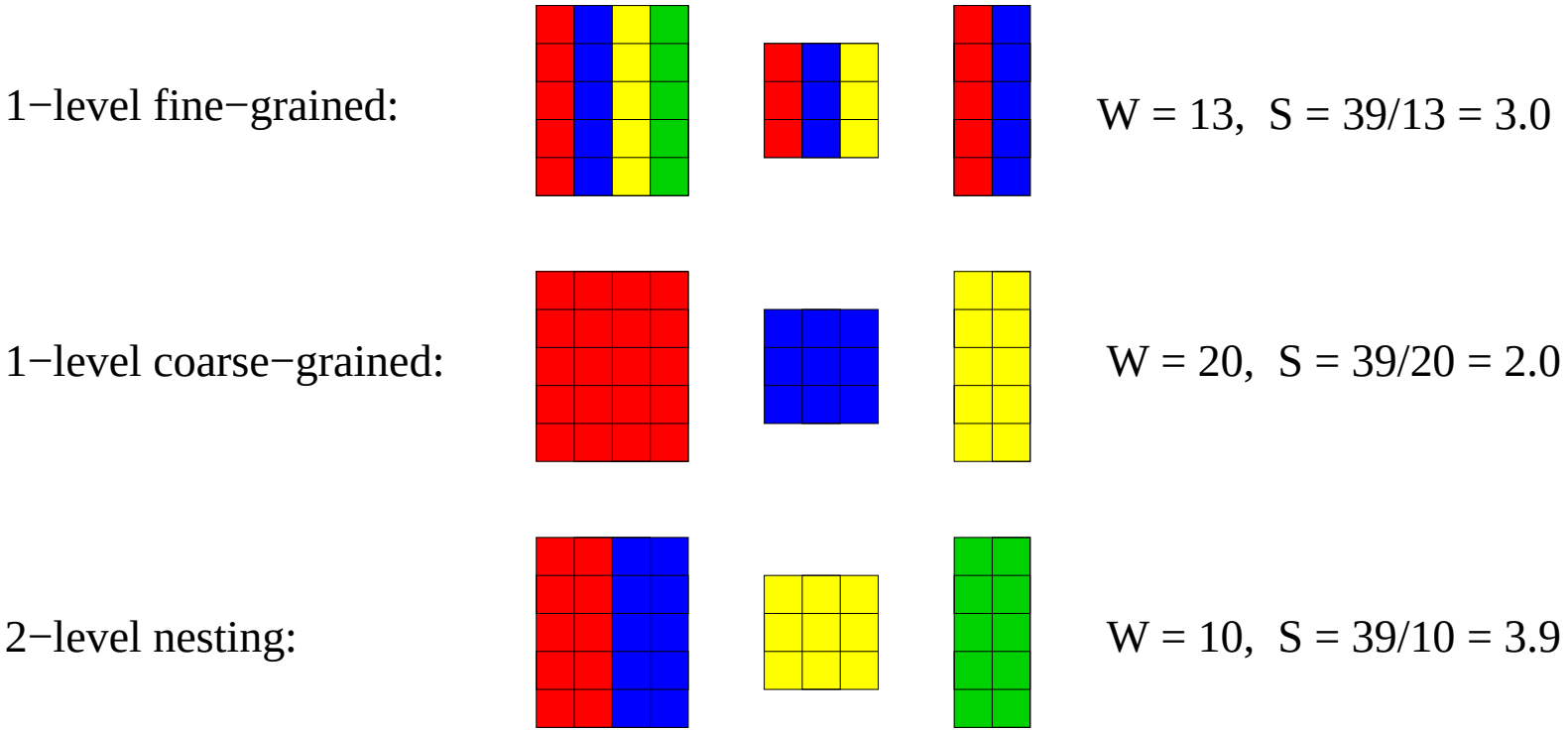


Figure 3. *AMR test case: Illustration of parallel approaches*

Overview

- Introduction
- Experiments - presentation
- Load balancing
- Implementation
- Experiments - results
- OpenMP
- Conclusions

Load balance

A distribution algorithm will be presented that allocates P threads to N tasks.

Two extreme cases:

1. All tasks need at least one thread ($P > N$)
2. All threads do at least one task ($P < N$)

Algorithm 1: Distribution of threads to tasks

/ This algorithm finds a distribution of the work of N tasks to P threads, for $P \geq N$. The weight of task i is denoted by w_i, while the number of threads distributed to task i is given by p_i. */*

[$p_{1:N}$] = **Distribute1**($N, P, w_{1:N}$)

$p_{1:N} = 1$;

for j = $N+1$, P

Find k such that $\frac{w_k}{p_k} = \max_{i=1,\ldots,N}(\frac{w_i}{p_i})$;

$p_k = p_k + 1$;

end for

Load balance: Case 2

Algorithm 2: Distribution of tasks to threads
(Sorted best fit for bin-packing in fixed number of bins)

/ This algorithm finds a distribution of the work of P threads to N tasks. The total amount of work allocated to thread i is given by t_i.*

[$t_{1:P}$] = **Distribute2**($N, P, w_{1:N}$)

Sort the tasks such that $w_i \geq w_j \quad \forall i < j$ and $i, j \in [1, N]$;

$t_{1:P} = w_{1:P}$;

for j = $P + 1$, N;

  Find k such that $t_k = \min_{i=1,\ldots,P}(t_i)$;

  $t_k = t_k + w_j$;

end for

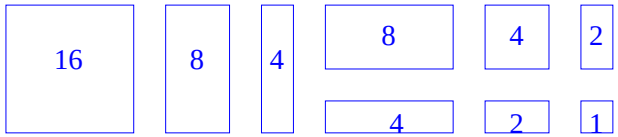
Load balance: Case 1+2

Algorithm 3: Combined algorithm

/ This algorithm finds a distribution of the work of N tasks to P threads.*
The tasks are divided in 2 sets, $\mathcal{L}$ ("large" tasks) and $\mathcal{S}$ ("small" tasks),
where one or more threads are working on the same task in $\mathcal{L}$,
and where each thread has one or more tasks to work on in $\mathcal{S}$.

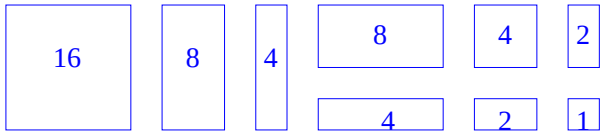
$[\mathcal{L}, \mathcal{S}, p_{1:N_\mathcal{L}}, t_{1:P_\mathcal{S}}]$ = **Distribute_All**$(N, P, w_{1:N})$

Load balance: Combined alg. applied to Wavelet TC

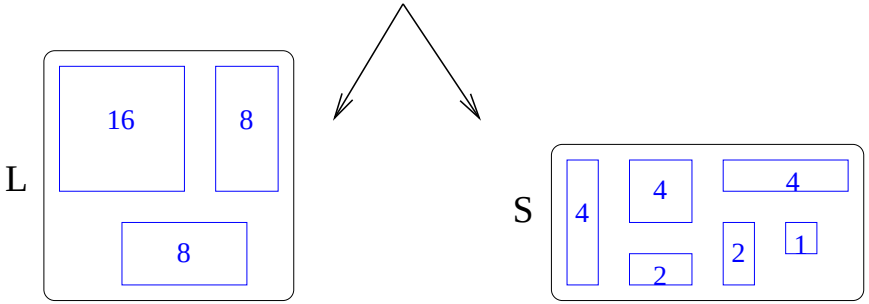


W = 49 P = 9

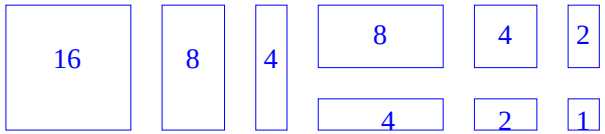
Load balance: Combined alg. applied to Wavelet TC



W = 49 P = 9 w = W/P = 5.4



Load balance: Combined alg. applied to Wavelet TC

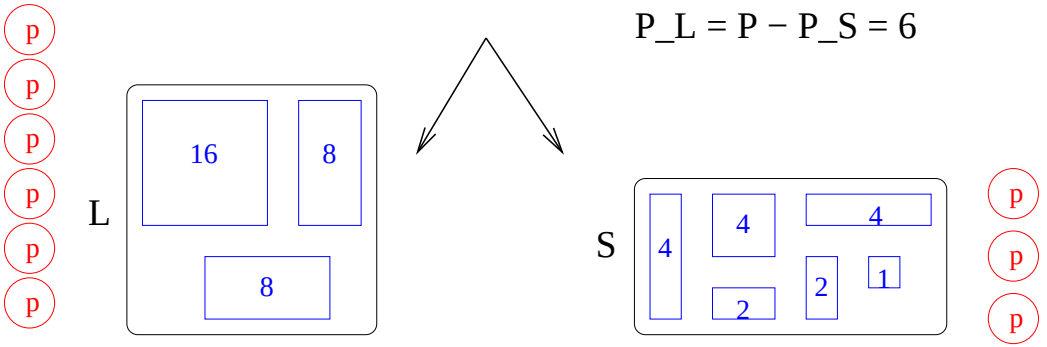


W = 49 P = 9 w = W/P = 5.4

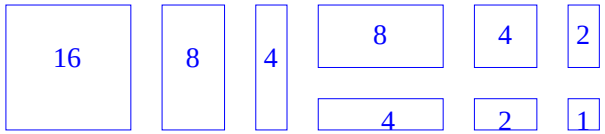
W_S = 4+4+4+2+2+1 = 17

P_S = int(W_S / w) = int(3.12) = 3

P_L = P − P_S = 6



Load balance: Combined alg. applied to Wavelet TC

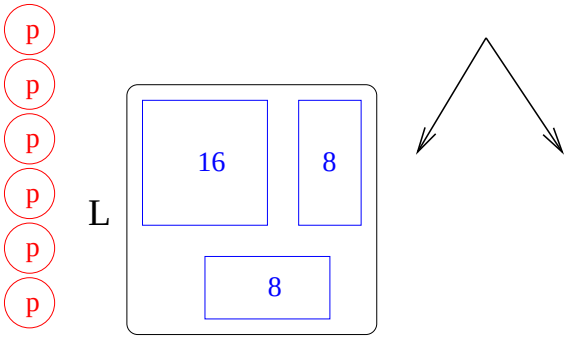


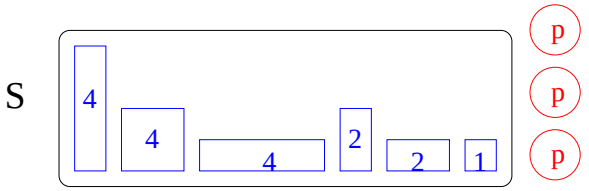
W = 49 P = 9 w = W/P = 5.4

W_S = 4+4+4+2+2+1 = 17

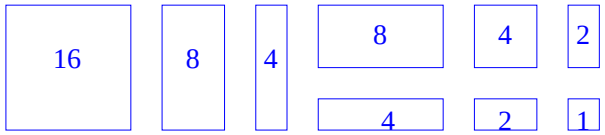
P_S = int(W_S / w) = int(3.12) = 3

P_L = P − P_S = 6





Load balance: Combined alg. applied to Wavelet TC

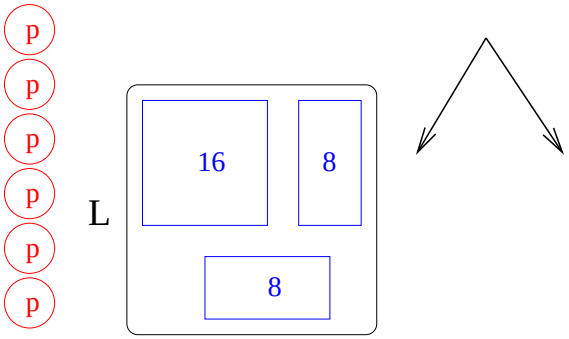


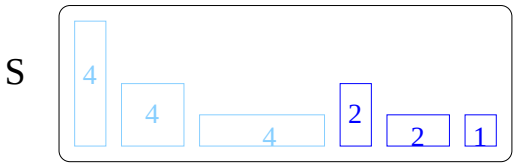
W = 49 P = 9 w = W/P = 5.4

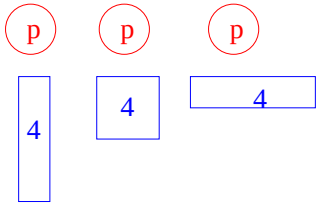
W_S = 4+4+4+2+2+1 = 17

P_S = int(W_S / w) = int(3.12) = 3

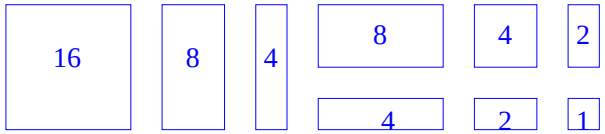
P_L = P − P_S = 6







Load balance: Combined alg. applied to Wavelet TC

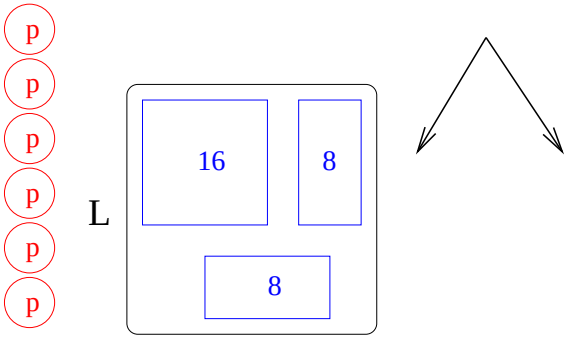


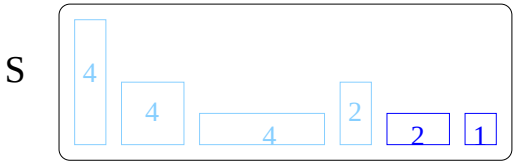
W = 49 P = 9 w = W/P = 5.4

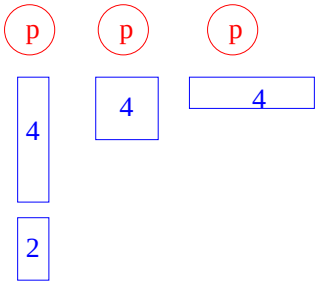
W_S = 4+4+4+2+2+1 = 17

P_S = int(W_S / w) = int(3.12) = 3

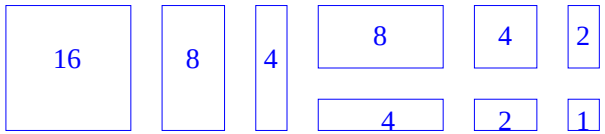
P_L = P − P_S = 6







Load balance: Combined alg. applied to Wavelet TC

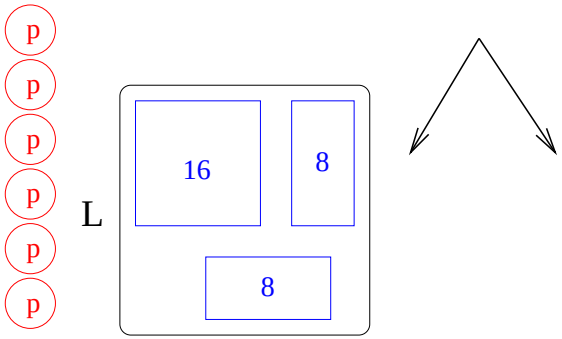


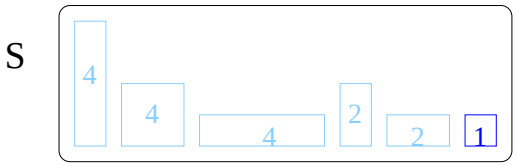
W = 49 P = 9 w = W/P = 5.4

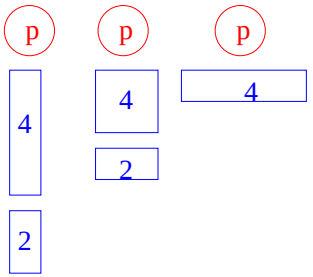
W_S = 4+4+4+2+2+1 = 17

P_S = int(W_S / w) = int(3.12) = 3

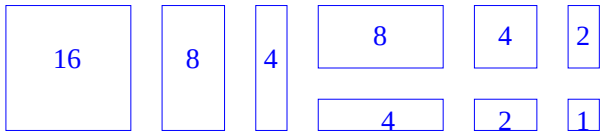
P_L = P − P_S = 6







Load balance: Combined alg. applied to Wavelet TC

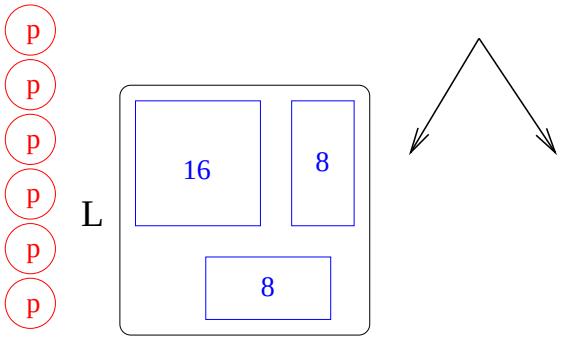


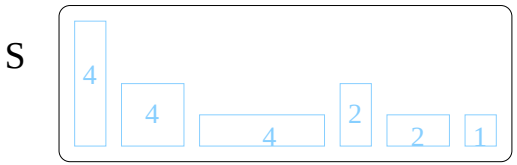
W = 49 P = 9 w = W/P = 5.4

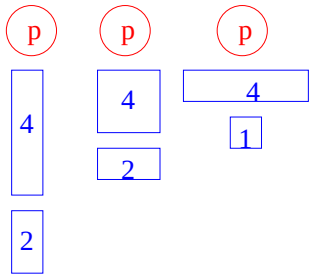
W_S = 4+4+4+2+2+1 = 17

P_S = int(W_S / w) = int(3.12) = 3

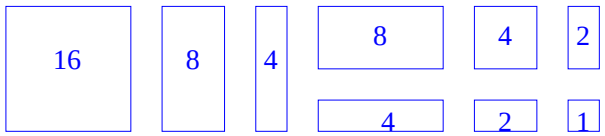
P_L = P − P_S = 6







Load balance: Combined alg. applied to Wavelet TC

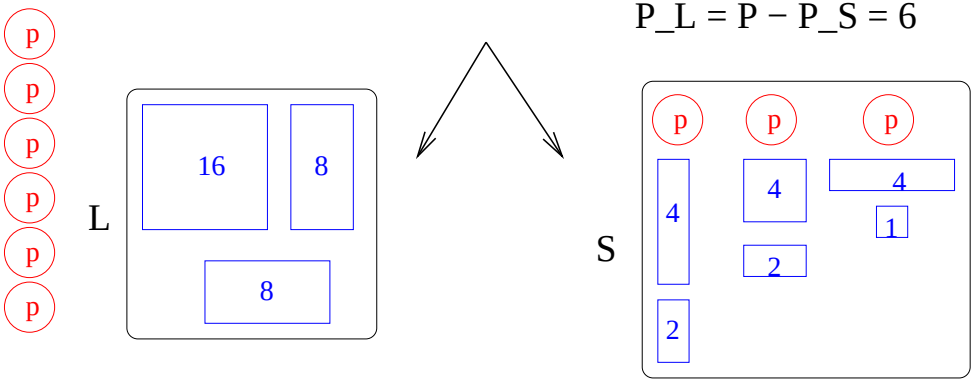


W = 49 P = 9 w = W/P = 5.4

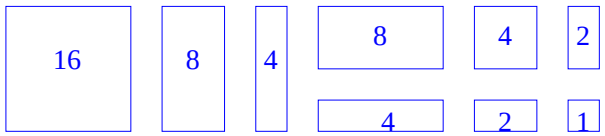
W_S = 4+4+4+2+2+1 = 17

P_S = int(W_S / w) = int(3.12) = 3

P_L = P − P_S = 6



Load balance: Combined alg. applied to Wavelet TC

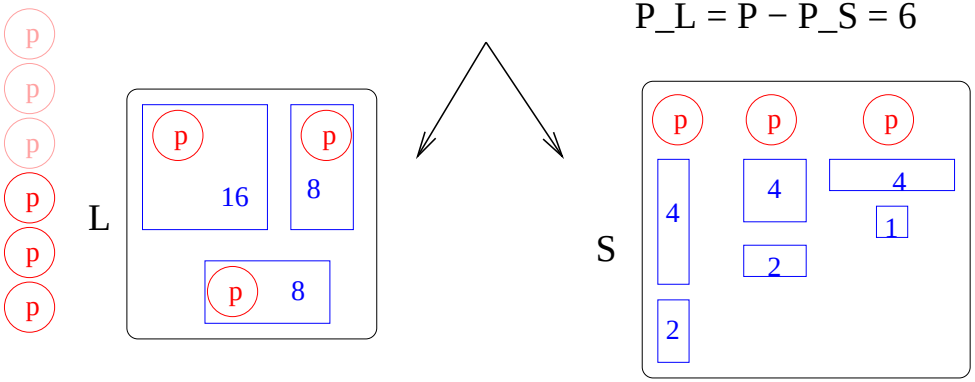


W = 49 P = 9 w = W/P = 5.4

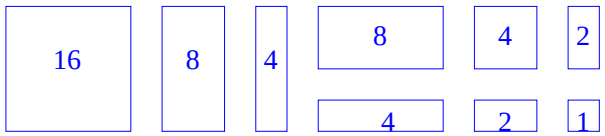
W_S = 4+4+4+2+2+1 = 17

P_S = int(W_S / w) = int(3.12) = 3

P_L = P − P_S = 6



Load balance: Combined alg. applied to Wavelet TC

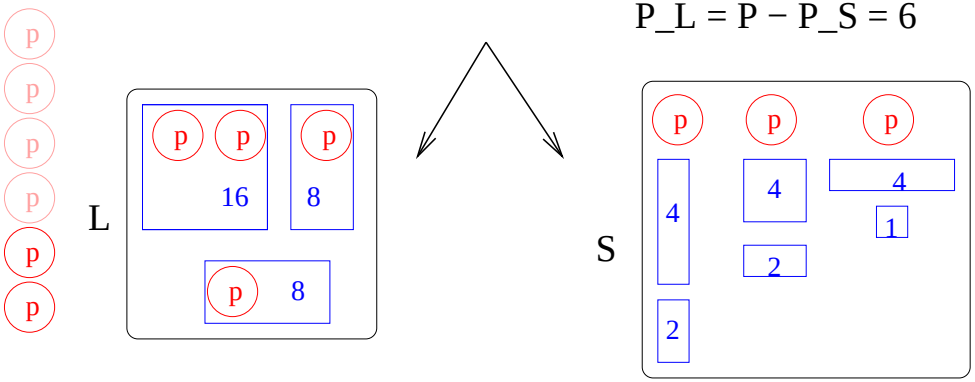


W = 49 P = 9 w = W/P = 5.4

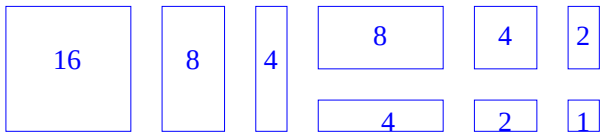
W_S = 4+4+4+2+2+1 = 17

P_S = int(W_S / w) = int(3.12) = 3

P_L = P − P_S = 6



Load balance: Combined alg. applied to Wavelet TC

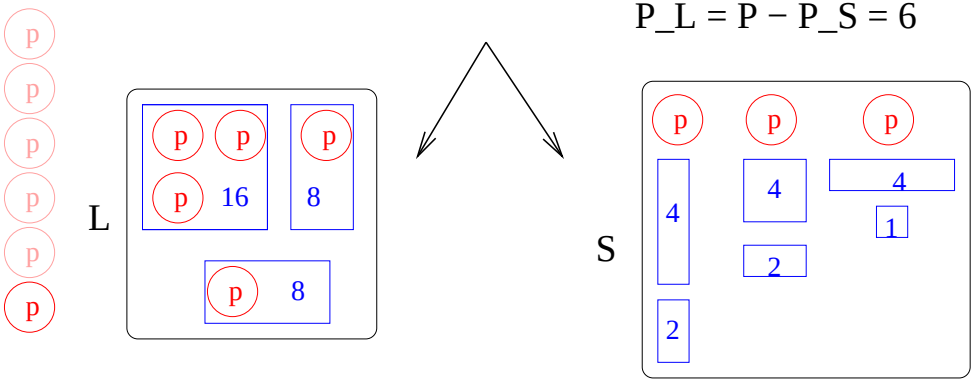


W = 49 P = 9 w = W/P = 5.4

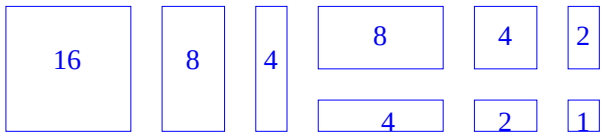
W_S = 4+4+4+2+2+1 = 17

P_S = int(W_S / w) = int(3.12) = 3

P_L = P − P_S = 6



Load balance: Combined alg. applied to Wavelet TC

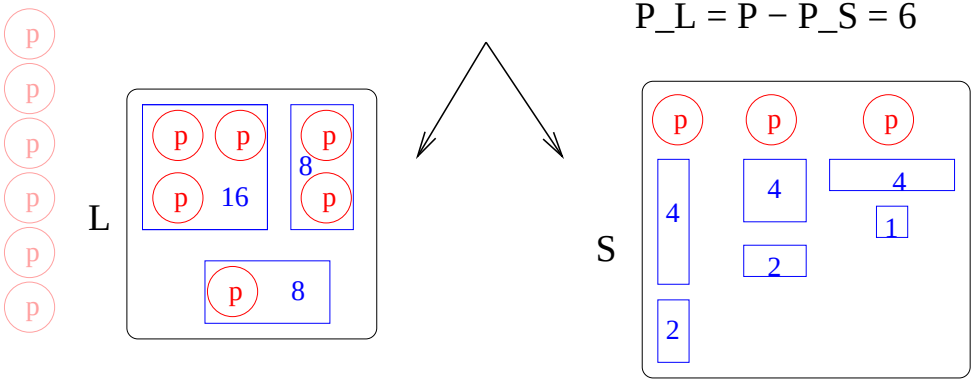


W = 49 P = 9 w = W/P = 5.4

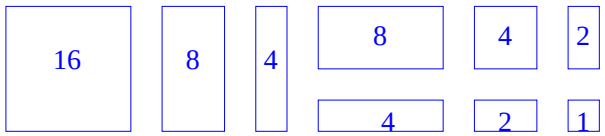
W_S = 4+4+4+2+2+1 = 17

P_S = int(W_S / w) = int(3.12) = 3

P_L = P − P_S = 6



Load balance: Combined alg. applied to Wavelet TC

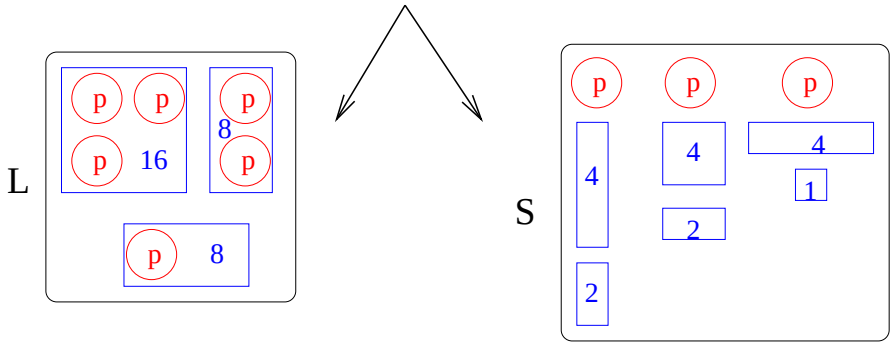


W = 49 P = 9 w = W/P = 5.4

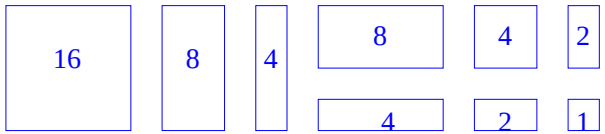
W_S = 4+4+4+2+2+1 = 17

P_S = int(W_S / w) = int(3.12) = 3

P_L = P − P_S = 6



Load balance: Combined alg. applied to Wavelet TC

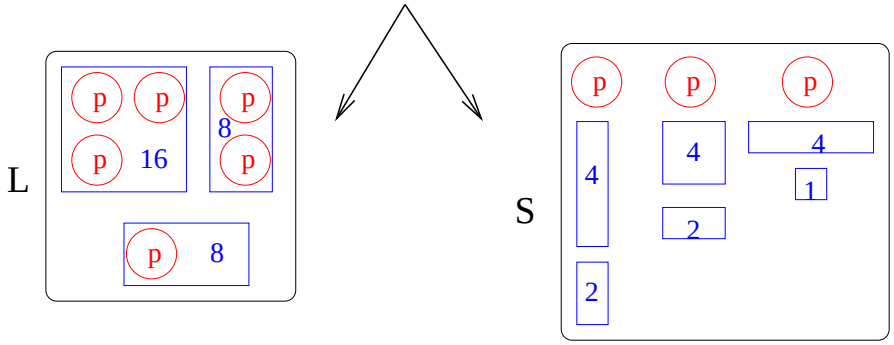


W = 49 P = 9 w = W/P = 5.4

W_S = 4+4+4+2+2+1 = 17

P_S = int(W_S / w) = int(3.12) = 3

P_L = P − P_S = 6



Max work = 8 Speedup = W/8 = 6.1

Load balance: Case 2

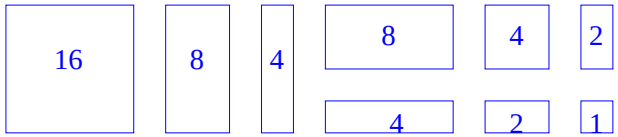
Alternative to Algorithm 2:

Algorithm 2b: Distribution of tasks to threads
(Sorted best fit for bin-packing in fixed sized bins)

/ This algorithm finds a distribution of N tasks to P threads.*
The maximum amount of work to be allocated to any thread is $\bar{w}$.
The total amount of work allocated to thread i is given by t_i.

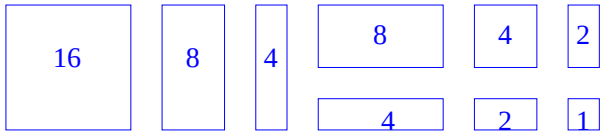
$[P, t_{1:P}]$ = **Distribute2b**$(N, \bar{w}, w_{1:N})$

Load balance: Combined alg. applied to Wavelet TC

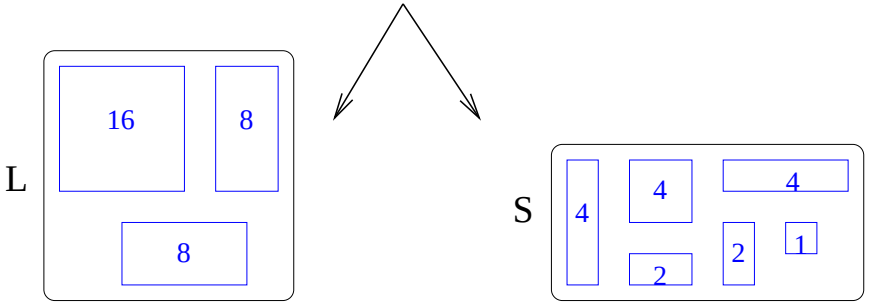


W = 49 P = 9

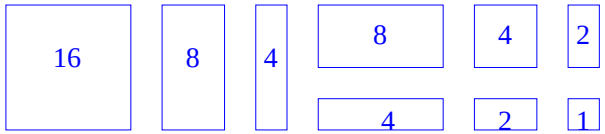
Load balance: Combined alg. applied to Wavelet TC



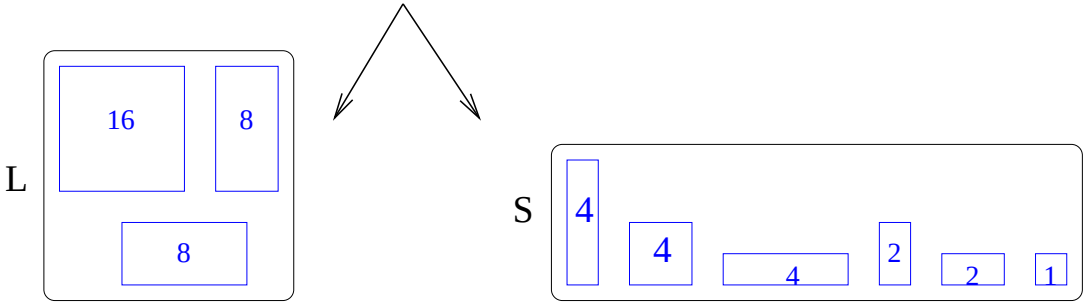
W = 49 P = 9 w = W/P = 5.4



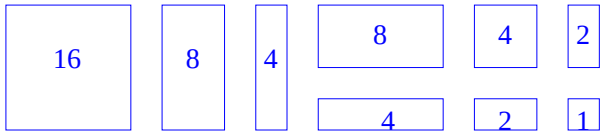
Load balance: Combined alg. applied to Wavelet TC



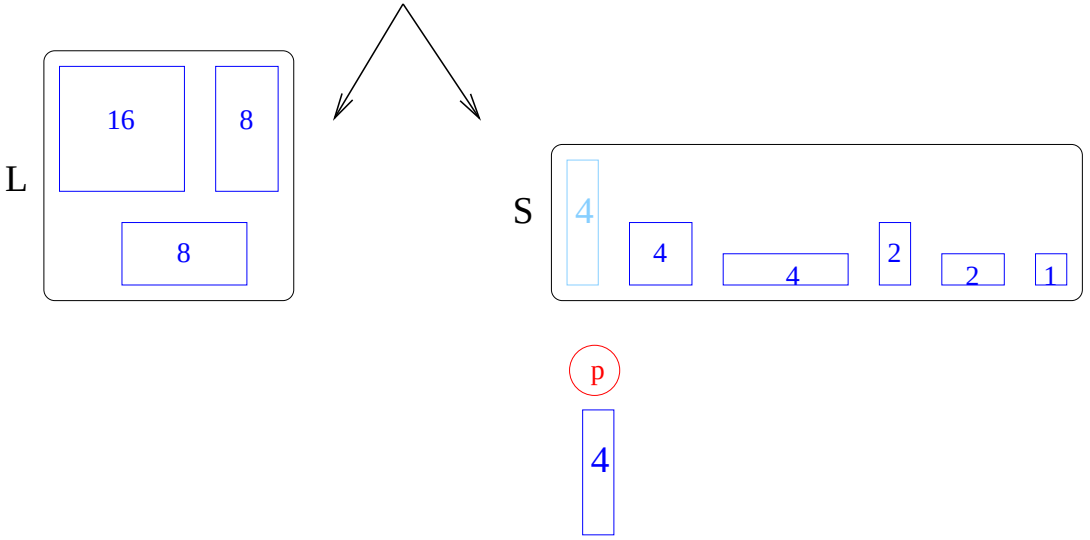
W = 49 P = 9 w = W/P = 5.4



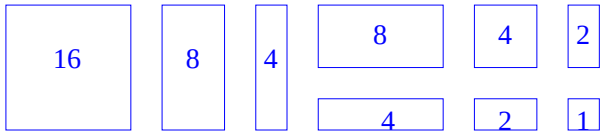
Load balance: Combined alg. applied to Wavelet TC



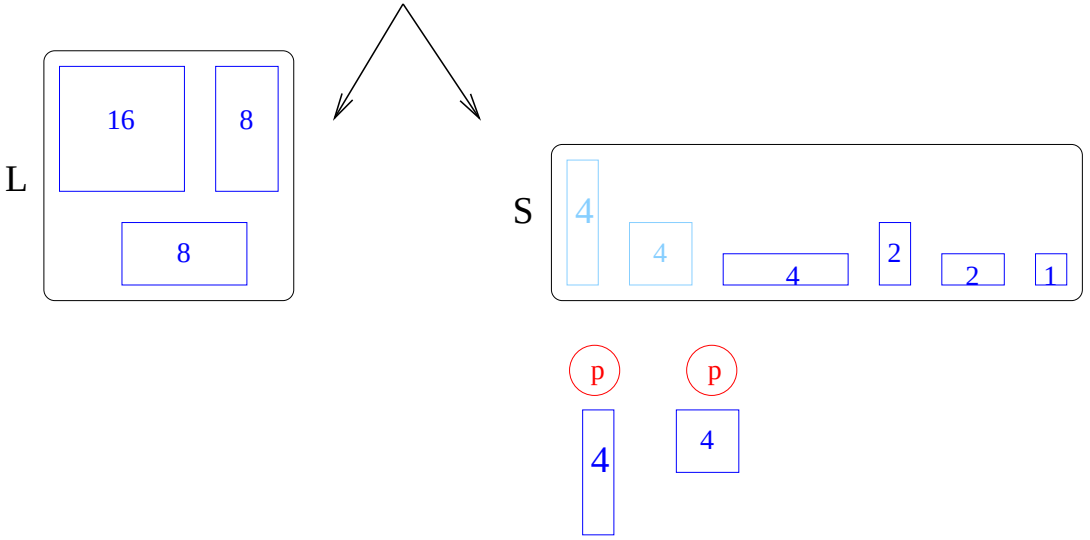
W = 49 P = 9 w = W/P = 5.4



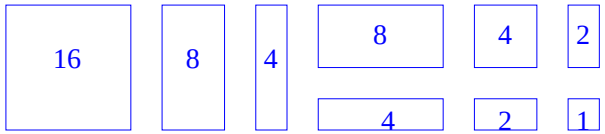
Load balance: Combined alg. applied to Wavelet TC



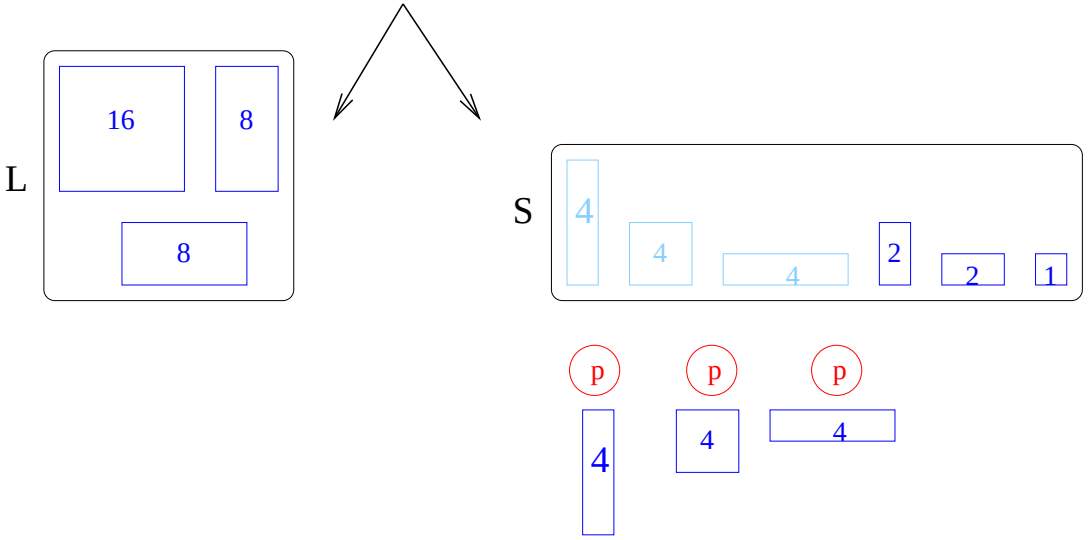
W = 49 P = 9 w = W/P = 5.4



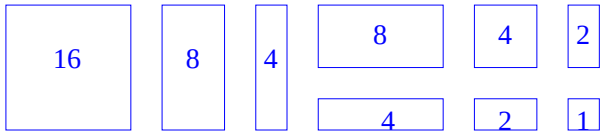
Load balance: Combined alg. applied to Wavelet TC



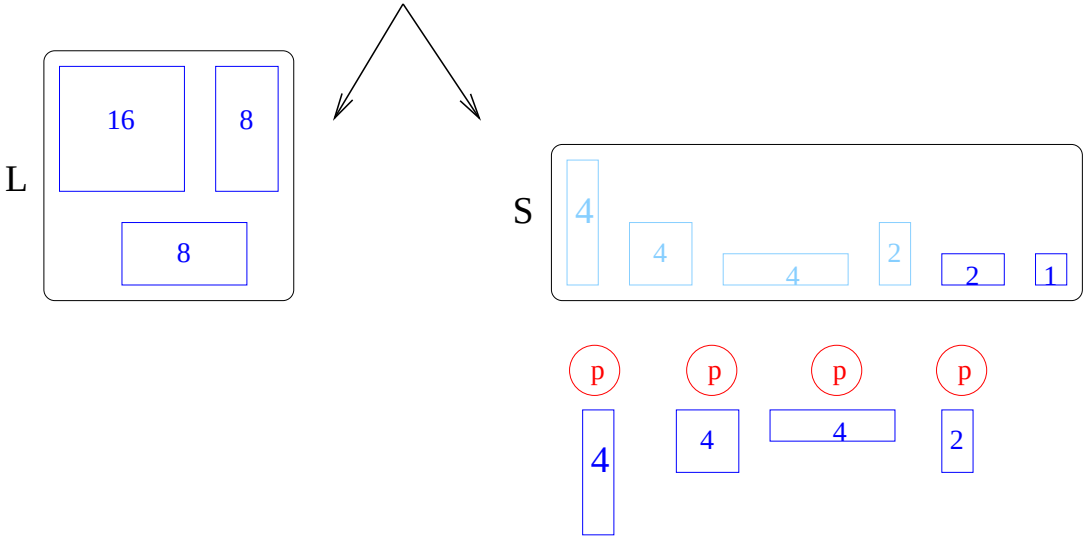
W = 49 P = 9 w = W/P = 5.4



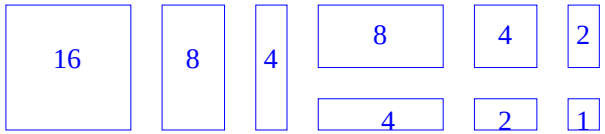
Load balance: Combined alg. applied to Wavelet TC



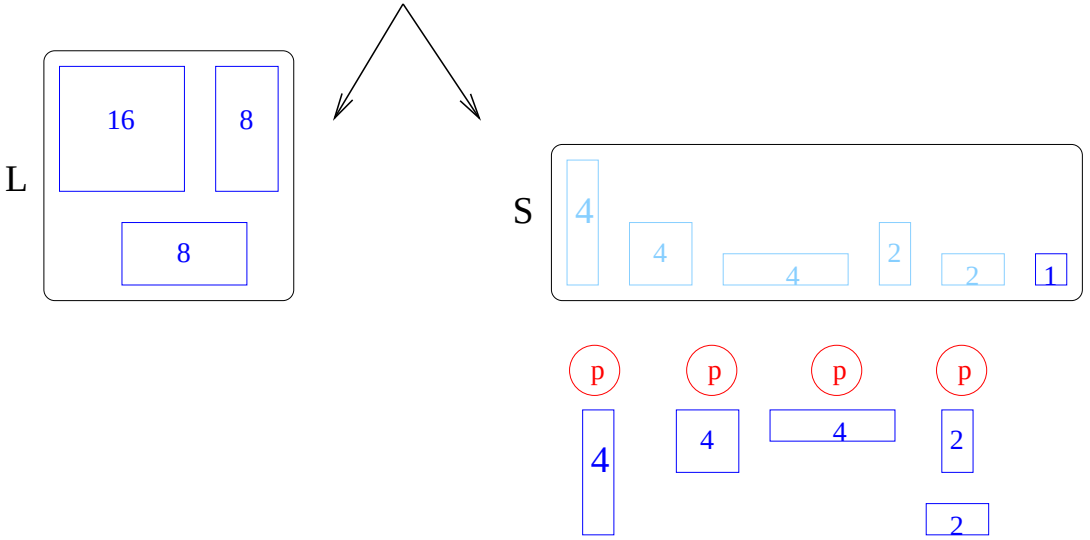
W = 49 P = 9 w = W/P = 5.4



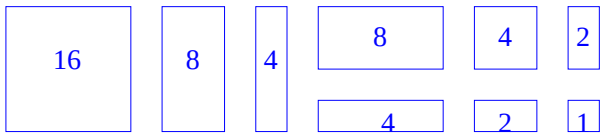
Load balance: Combined alg. applied to Wavelet TC



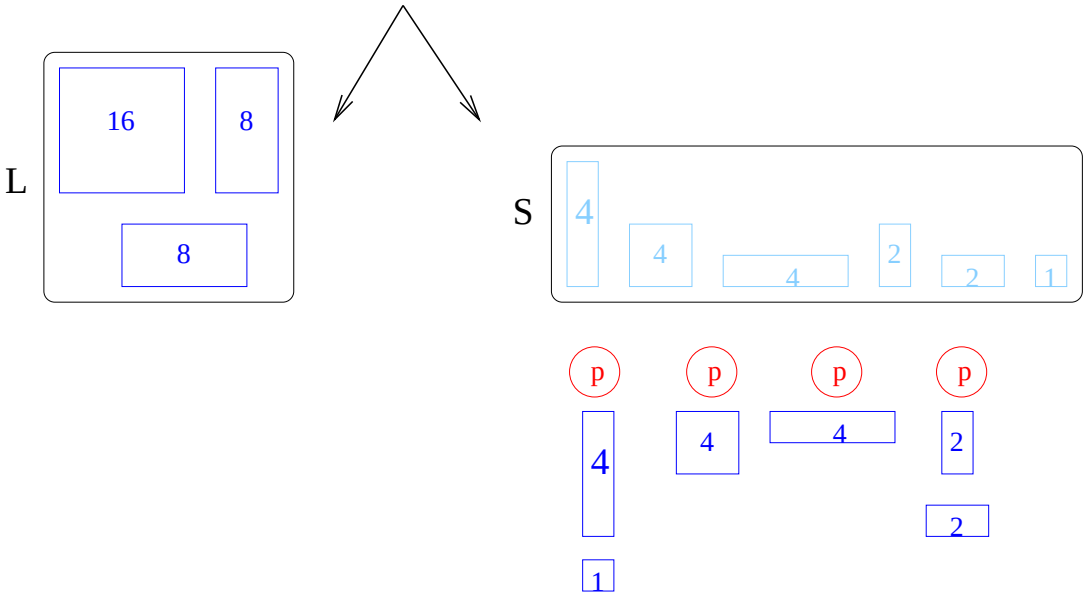
W = 49 P = 9 w = W/P = 5.4



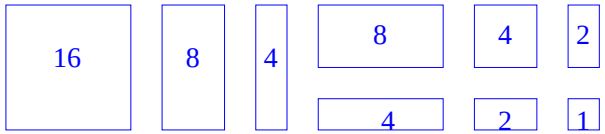
Load balance: Combined alg. applied to Wavelet TC



W = 49 P = 9 w = W/P = 5.4

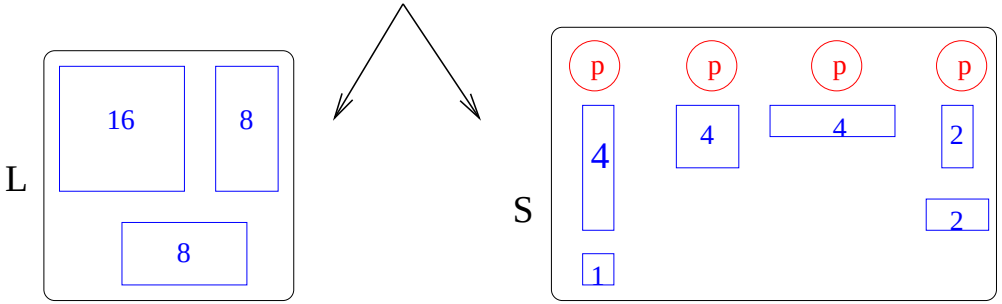


Load balance: Combined alg. applied to Wavelet TC

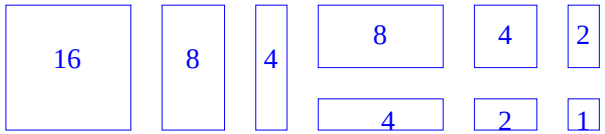


W = 49 P = 9 w = W/P = 5.4

P_S = 4

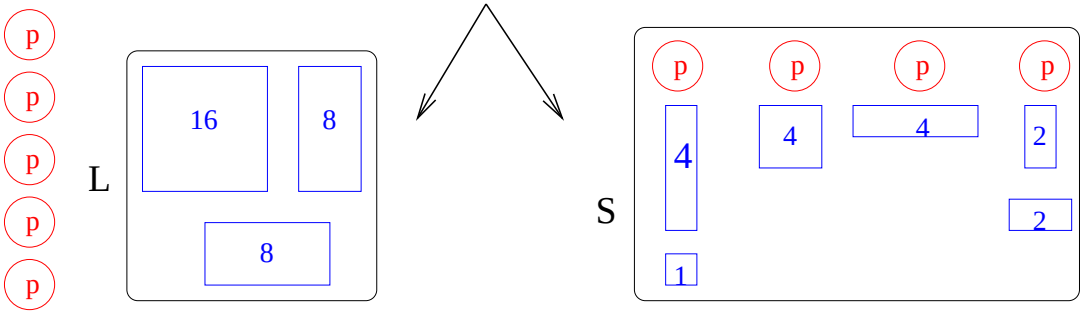


Load balance: Combined alg. applied to Wavelet TC

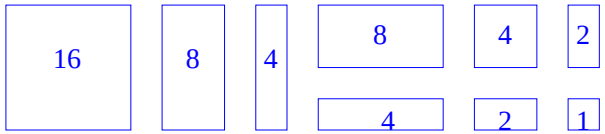


W = 49 P = 9 w = W/P = 5.4

P_S = 4 P_L = P − P_S = 5

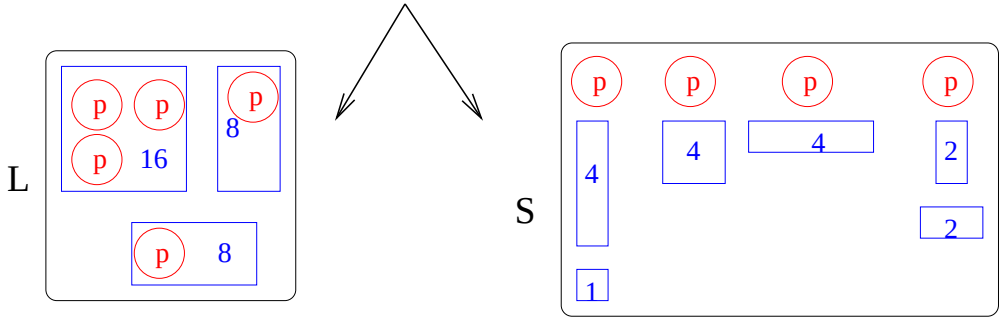


Load balance: Combined alg. applied to Wavelet TC

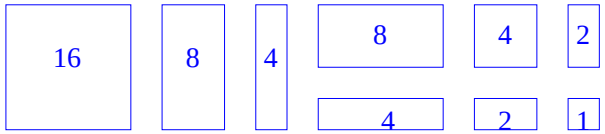


W = 49 P = 9 w = W/P = 5.4

P_S = 4 P_L = P − P_S = 5

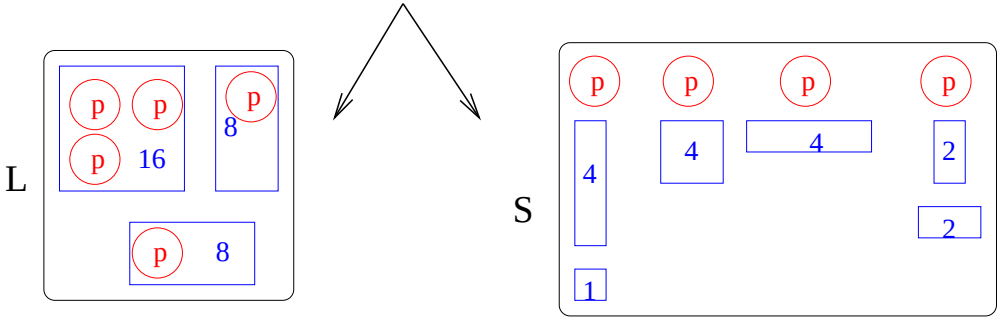


Load balance: Combined alg. applied to Wavelet TC



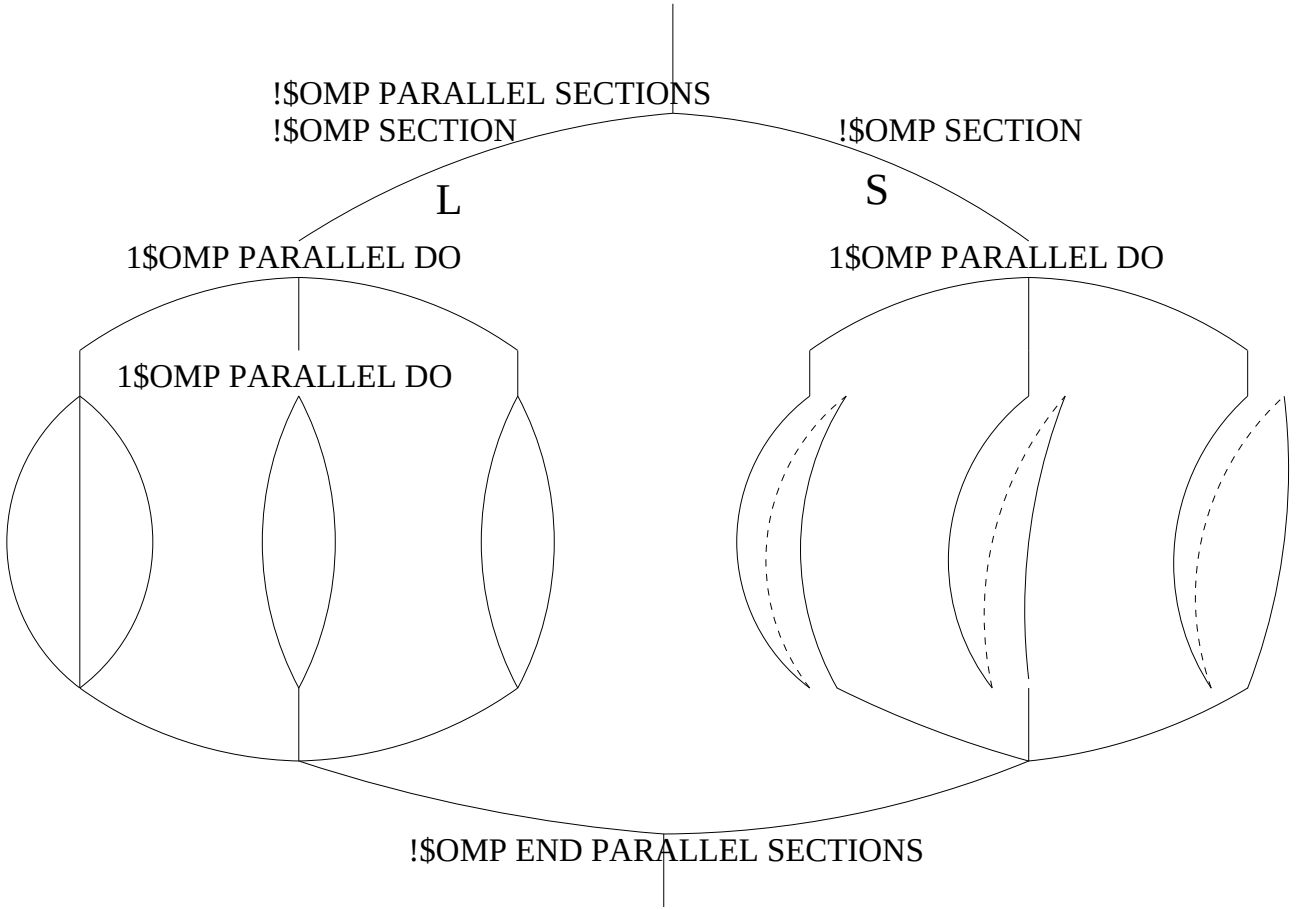
W = 49 P = 9 w = W/P = 5.4

P_S = 4 P_L = P − P_S = 5



Max work = 8 Speedup = W/8 = 6.1

Implementation: Combined alg.



Implementation: Case 1

According to the OpenMP standard, nested parallelization in a double Fortran loop can be achieved by the following directives:

```
!$OMP PARALLEL DO PRIVATE(i) NUM_THREADS(N)
      do i = 1, N
!$OMP PARALLEL DO PRIVATE(j), SHARED(i) NUM_THREADS(p(i))
         do j = 1, w(i)
            < WORK(i,j) >
         end do
      end do
```

Implementation: Case 1

Unfortunately, OpenMP-compliant implementations are allowed to serialize nested parallel regions, even when nested parallelism is enabled.

Still most vendors have chosen to serialize parallel regions.

Fortunately, OpenMP allows a more low level work distribution, where the programmer explicitly assign work to threads. This technique can be used to obtain nested parallelism.

Implementation: Case 1

Nested parallelism achieved by explicit assignment of work to threads:

```
      call algorithm1
!$OMP PARALLEL PRIVATE(thread,i,j)
      thread = OMP_GET_THREAD_NUM()
      i = mytask(thread)
      do j = jbegin(thread), jend(thread)
         < WORK(i,j) >
      end do
!$OMP END PARALLEL
```

Implementation: Case 1

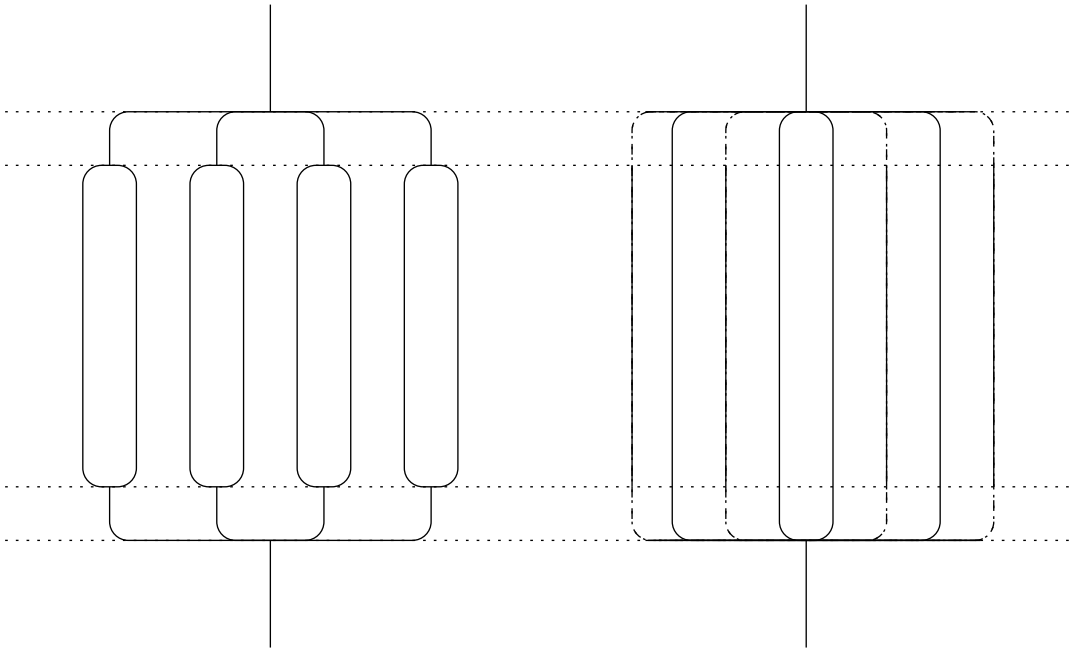


Figure 4. *Case 1: Illustration of 2-level parallelism.*

Experiments - results

- A wavelet based data compression routine
  - having a static number of tasks
- An adaptive mesh refinement (AMR) application
  - having a dynamically changing number of tasks

Experiments - results: Wavelet test case

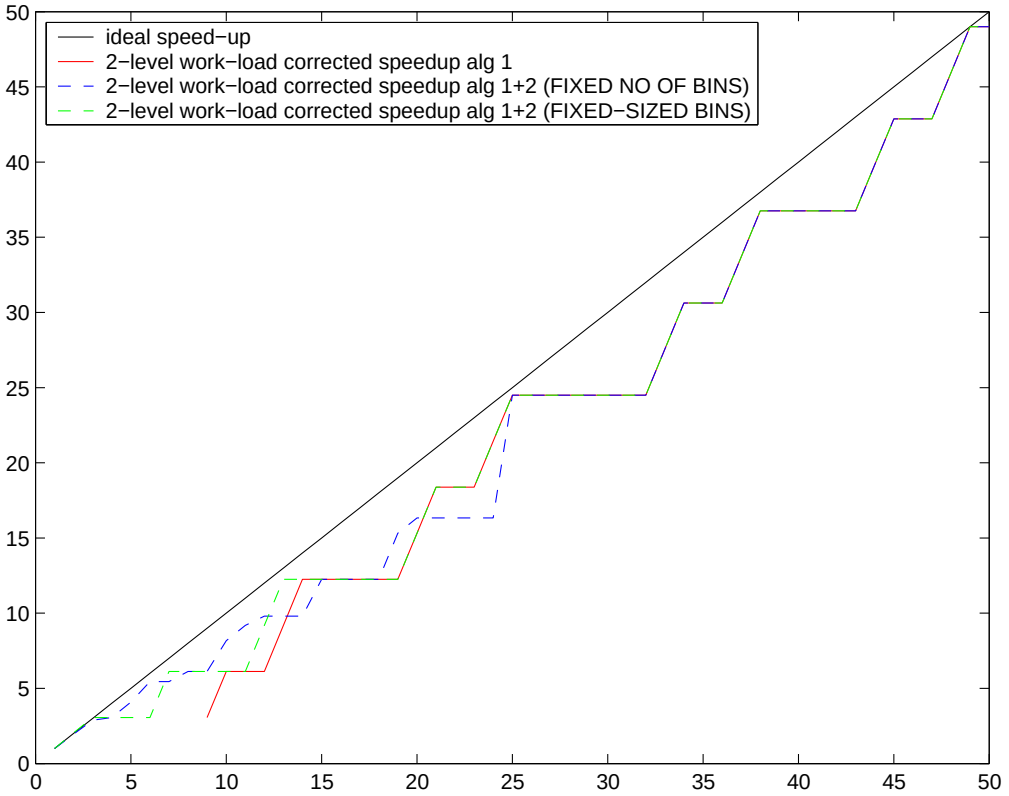


Figure 5. *Wavelet test case: Theoretical speedup*

Experiments - results: Wavelet test case

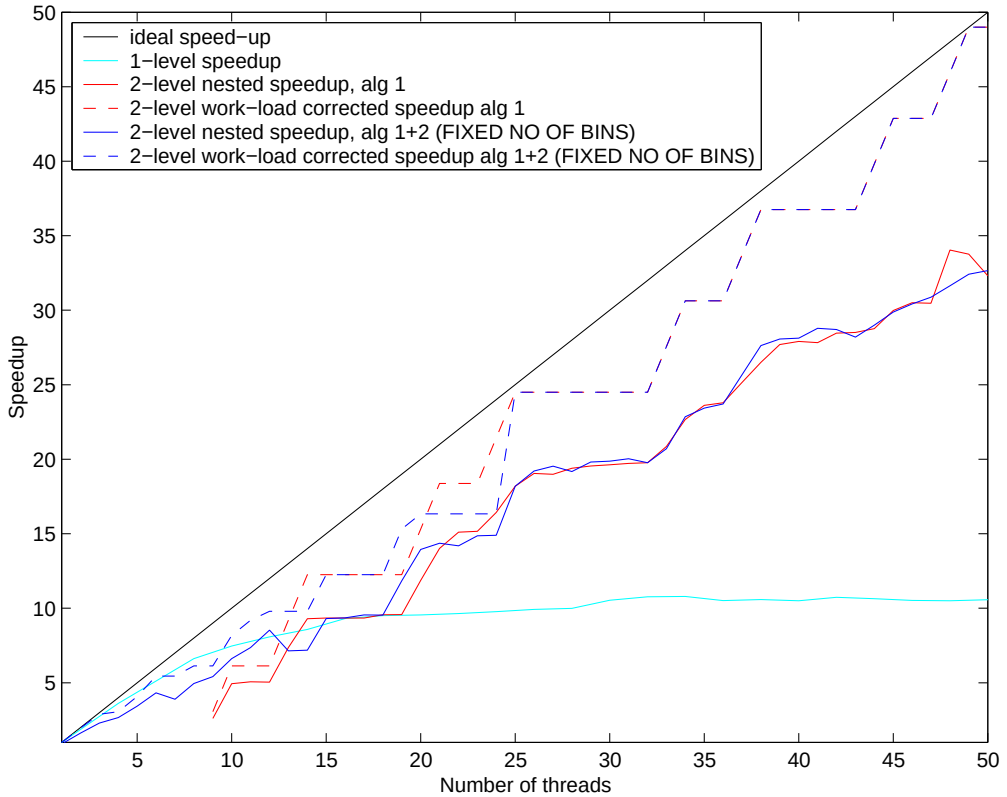


Figure 6. *Wavelet test case: Obtained speedup*

Experiments - results: An AMR application

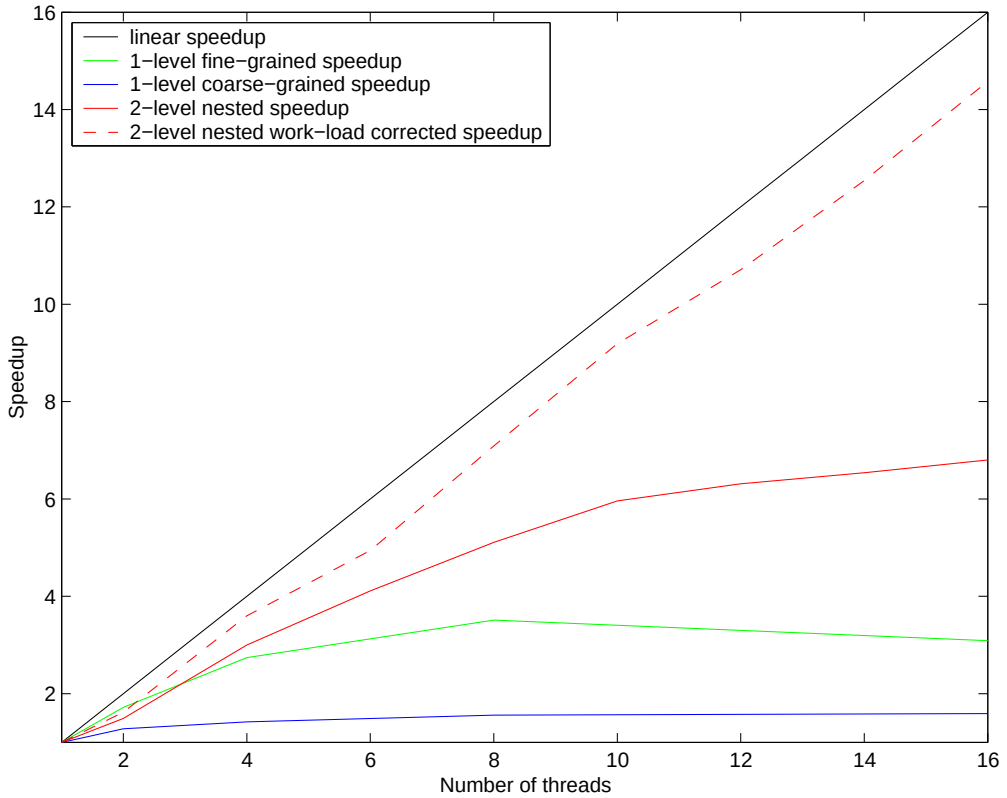


Figure 7. *AMR test case: Obtained speedup*

Implementation in OpenMP

Even when nesting is implemented on a system,
we miss the directive

```
!$OMP TEAMPRIVATE
```

corresponding to `!$OMP THREADPRIVATE`.

Conclusions

- An algorithm for allocation of threads to tasks, which cover all possible mixes of tasks, and works for an arbitrary number of threads, is presented.
- Parallelization by explicit thread programming is more time consuming than directive based parallelization. Directive based parallelization is therefore preferred when nesting is available on the system.
- However, nested parallelism obtained by explicit thread programming is shown to be a good alternative when directive based nesting is not available.
- Nested parallelism pays off when the problem consists of multiple layers of tasks, both for a static and a dynamic number of tasks.