Nested parallelism: Allocation of processors to tasks and OpenMP implementation

Ragnhild Blikberg and Tor Sørevik
Department of Informatics
University of Bergen
Norway
Outline of the talk

1. Introduction
2. Motivation
3. Algorithm
4. Implementation in OpenMP
5. Experiments
6. Shortcomings of the OpenMP directives
7. Conclusions and future work
1. Introduction

When is nested parallelism desirable?

Many computational problems have an outer-level of coarse grained parallelism, where the number of tasks are few, but where each task contains a large amount of work.

Each such outer-level task might itself be a parallel task of more fine grained parallelism.

Problems like this invites to the use of multi-level parallelism, or nested parallelism.
2. Motivation

Motivating example:

\[
\begin{align*}
\text{do } i & = 1, 4 \\
\text{do } j & = 1, w(i) \\
& \quad \text{< work >} \\
& \quad \text{end do; end do}
\end{align*}
\]

The \( i \)-loop within each patch is data-independent.

1-level parallelization:

A) Parallelizing the \( i \)-loop will limit the number of processors to 4.
B) Parallelizing the \( j \)-loop, the number of maximum possible processors will vary from 2 to 36.
Motivation, general case

Suppose: \( w_i = \) computational cost of parallel part in task \( i \)
\( s_i = \) computational cost of sequential part in task \( i \)

Sequential runtime for \( N \) tasks:

\[
T_1 = \sum_{i=1}^{N} (w_i + s_i) = W + S
\]

Applying only inner-level parallelization, the runtime on \( P \) processors will be

\[
T_P = \sum_{i=1}^{N} \left( \frac{w_i}{P} + s_i \right) = \frac{W}{P} + S
\]
Suppose: \( p_i \) = number of processors associated with task \( i \)

For nested parallelism (NP) the runtime on \( P \) processors will be

\[
T_{NP} = \max_i \left( \frac{w_i}{p_i} + s_i \right)
\]

Assume that the load balance is perfect for the \( N \) outer tasks: \( \frac{w_i}{p_i} = \frac{W}{P} \)

If in addition \( s_1 = s_2 = \ldots = s_N = \frac{S}{N} \), the runtime for the optimal 2-level parallelization on \( P \) processors will be

\[
T_{NP} = \frac{W}{P} + \frac{S}{N} < T_P = \frac{W}{P} + S
\]
Figure 1: Speed-up curves for idealized cases with $N = 10$ and $S = 0.01W$. 
3. Algorithm for allocating processors to tasks

**Symbols:**

\( N = \text{number of tasks} \)

\( P = \text{total number of processors} \)

\( w_i = \text{weight of task } i \)

\( p_i = \text{number of processors allocated to task } i \)

Suppose \( P > N \).

**Problem:** Find an optimal distribution of processors to tasks such that

\[
\max_{i} \left( \frac{w_i}{p_i} \right) \text{ is minimized and } \sum_{i=1}^{N} p_i = P.
\]
Algorithm: Distribution of processors to tasks

for $i = 1, N$
    $p_i = 1$;
end for

for $j = N + 1, P$
    update heap such that
    \[
    \frac{w_1}{p_1} \geq \max \left( \frac{w_2}{p_2}, \ldots, \frac{w_N}{p_N} \right);
    \]
    $p_1 = p_1 + 1$;
end for

It can be proved that this algorithm is optimal, and that the complexity is $O((P - N)\log N)$. 
4. Implementation of nesting in OpenMP

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

```
 !$OMP PARALLEL DO  
   do i = 1,N  
!! $OMP PARALLEL DO  
      do j = 1,w(i)  
            < work >  
      end do  
!! $OMP END PARALLEL DO  
   end do  
!! $OMP END PARALLEL DO  
```
Manual nesting

-by explicit assignment of work to processors

call distribute

!$OMP PARALLEL DO PRIVATE(proc,i,j,...)
do proc = 1,P
   i = mytask(proc)
   do j = jbegin(i,proc), iend(i,proc)
      < work >
   end do
end do
!$OMP PARALLEL DO
Figure 2: 1-level and 2-level parallelization

1-level parallelism
- by 1 directive

2-level parallelism
- by 2 directives
- by 1 directive and changing the code

Figure 2: 1-level and 2-level parallelization
5. Experiments

5.1 Matrix multiplication

Originally:

\[
\begin{align*}
\text{do } & i = 1,N \\
& \text{do } j = 1,w(i) \\
& \quad \text{do } k = 1,m \\
& \quad \quad \text{do } l = 1,m \\
& \quad \quad \quad C(l,j,i) = A(l,k,i) \cdot B(k,j,i) + C(l,j,i) \\
\end{align*}
\]
Nested 2-level parallelization of matrix multiplication:

call distribute

!$OMP PARALLEL DO PRIVATE(proc,i,j,k,l)
do proc = 1,P
   i = mytask(proc)
do j = jbegin(i,proc), jend(i,proc)
do k = 1,m
do l = 1,m
   C(l,j,i) = A(l,k,i)*B(k,j,i) + C(l,j,i)
end do
end do
end do
Figure 2: Speed-ups on matrix multiplication for $N = 4$ (susped machine)
5.2 Data compression
- a more realistic case, used in an out-of-core earthquake simulator
- contains areas independent of each other
Theoretical speedup for 2-level nested parallelization:

\[ S = \frac{\sum_{i=1}^{N} (w_i + s_i)}{\max_i \left( \frac{w_i}{p_i} + s_i \right)} \]
Figure 4: Speed-ups for data compression and $N = 9$ (dedicated machine)
6. Shortcomings of the OpenMP directives

- We miss a clause to the `!$OMP PARALLEL` directive telling how many processes which shall be created in the parallel region.

  In the draft of OpenMP 2.0 it is proposed such a clause to the parallel regions directives.

- In the draft of OpenMP 2.0 it is unfortunately proposed that nested parallelism should still be implementation dependent.

- The OpenMP Nanos compiler shows that this is feasible!
7. Conclusions and future work

• Theoretical and practical examples show the advantage of multilevel parallelism.

• A simple load balancing algorithm has been presented, and it is proved to give the optimal allocation of processors to tasks.

• There are shortcomings of OpenMP 1.0 nesting directives.

• Nesting can be implemented in OpenMP using explicit thread programming.

• We are very unhappy with the fact that serializing nested parallelism is still compliant with the OpenMP spec.

• Future work: nesting in AMR