"No-Compile-Time Knowledge" Distribution of Finite Element Computations on Multiprocessors

J. Erhel  M. Ilahad  T. Priol

INRIA
IRISA, Campus de Beaulieu
35042 Rennes Cedex, France

Abstract

This paper addresses partitioning and scheduling of irregular loops arising in finite element computations on unstructured meshes. Target computers are Distributed Memory Parallel Computers that provide a global address space. We introduce the concept of "Conditioned Iterations Loop" which distributes the iterations dynamically according to a runtime condition. This technique is improved by a learning approach. It is also combined with a renumbering of the mesh to improve data locality. Execution times on KSR1 with up to 64 processors on a model loop show good performances.

1 Introduction

This paper addresses partitioning and scheduling of irregular computations in finite element methods. Many authors, as in [1, 2], have considered the implementation of finite element irregular loops on parallel MIMD computers. Here we address their implementation using a shared virtual memory. We are still faced to the problem of writing conflicts and data locality (here page faults). Moreover, the false sharing problem arises. Data locality can be improved by a renumbering as in [3]. Writing conflicts can be resolved by introducing a local finite element vector in each processor [4]. Here we introduce a new concept, called Conditioned Iterations Loop (CIL) [5] combined with a learning technique. Though our schemes are similar to the inspector/executor process in PARTI, it differs in several points due to the shared virtual memory environment. These execution schemes are integrated into a parallel programming environment currently under development with a code generator called Fortran-S [6]. In order to show the efficiency of our execution schemes, we apply them to a model loop which simulates finite element computations.

2 Conditioned Iterations Loop (CIL)

Let us consider for example partial differential equations with a time derivative. If we apply an explicit time discretisation, we then get an outer loop on time steps where each iteration contains computations of right-hand sides. These are approximated by a finite element discretisation, leading to irregular computations. We consider the following model loop in the case of triangular elements, with three nodes by element:

ALGORITHM : SCATTER-FEM

\[
\text{do } n=1,\text{ntimes} \\
\quad \text{do } j=1,\text{abel} \\
\quad \quad \text{tmp} = \ldots \\
\quad \quad S(La(j)) = S(La(j)) + \text{tmp} \\
\quad \quad S(Lb(j)) = S(Lb(j)) + \text{tmp} \\
\quad \quad S(Lc(j)) = S(Lc(j)) + \text{tmp} \\
\quad \text{endo} \\
\text{endo}
\]

We distribute dynamically the loop by introducing a condition such that if an iteration is executed on several processors, the condition is true on one and only one processor. We define a owner-based condition similar to the owner-computes rule. However, because our condition is virtual, it allows to balance the work. False sharing is automatically eliminated and, except for the start-up, write page faults are also suppressed. But this scheme CIL suffers from an overhead due to the inspection by all the processors. To improve CIL, we introduce a learning process which exploits the static indirection in the inner loop. It has a low memory and CPU cost thanks to the shared virtual memory environment. Experimental results show a dramatic improvement over the basic CIL, even with a small number of iterations ntimes.
In the context of finite element methods, we must take into account multiple writes into the same shared array. We design three schemes based on various conditions, all of them with a learning technique [7]. In the first scheme, denoted SLICE, the loop is sliced into three parts, requiring a scalar extension for the variable temp. This method reduces write page faults for the shared variable \( S \) but induces page migration for the computed \( tmp \). The main drawbacks of this method are the memory cost and the synchronisation points. Therefore we also design a second approach, called CLUSTER, which combines conditions and critical sections. The memory cost is very low here and synchronisation is reduced to a few elements. The efficiency depends on the overhead due to the locking operations and to the page faults occurring in the critical sections.

We design a third scheme with the same memory cost but with no page fault, at the price of redundancy in the computations. The idea is to compute \( tmp \) as soon as the processor owns at least one of the three pages.

The code generation of our CIL schemes is supported by Fortran-S thanks to user annotations. We define new directives to support CIL and learning [7]. We give below the sketch of the program:

```fortran
C$ann[Learn(1)]
Do t=1, ntimes
  C$ann[CILSTATIC,BLOCK])
  Do i=1,m
    S(L(i)) = S(L(i)) + ...
  EndDo
EndDo
```

3 Numerical Experiments

These three compilation schemes have been implemented and tested on a KSR1 computer, using the model loop, with indirections coming from various finite element meshes [7]. We studied two different renumberings of each mesh: either randomly mixed (with -m) or using a wavefront approach (with -r) to improve data locality. We have simulated the computation of \( tmp \) with either 10 or 100 floating-point operations. The number of time steps \( ntimes \) has been fixed to 100. We compare our CIL schemes to a distribution of iterations by blocks with critical sections, noted BLOCK. Results with 63 processors are given in Table 1 for a mesh with 207691 nodes, noted 200k. As could be expected, performances are better when the gather operation is sufficiently time-consuming. We obtain a significant improvement thanks to the renumbering of the mesh. If the mesh is not renumbered, our CIL schemes are more efficient than the basic BLOCK version.

We plan now to use these schemes in various complete scientific applications in order to measure there global efficiency. We want also to study more sophisticated renumberings based on mesh partitioning techniques in order to still improve data locality.

<table>
<thead>
<tr>
<th>10 flops</th>
<th>BLOCK</th>
<th>SLICE</th>
<th>OVERLAP</th>
<th>CLUSTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>200k+r</td>
<td>14.41</td>
<td>9.78</td>
<td>23.15</td>
<td>11.89</td>
</tr>
<tr>
<td>200k-m</td>
<td>6.57</td>
<td>13.99</td>
<td>36.45</td>
<td>7.29</td>
</tr>
<tr>
<td>100 flops</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200k+r</td>
<td>51.98</td>
<td>48.99</td>
<td>52.00</td>
<td>49.75</td>
</tr>
<tr>
<td>200k-m</td>
<td>21.65</td>
<td>35.51</td>
<td>25.29</td>
<td>22.65</td>
</tr>
</tbody>
</table>

References


