Improving Effective Bandwidth through Compiler Enhancement of Global Cache Reuse

Chen Ding
Department of Computer Science
University of Rochester
Rochester, NY

Ken Kennedy
Department of Computer Science
Rice University
Houston, TX
Introduction

• Because the memory wall

• Two-step strategy:
  • Global computation fusion
    • Fuses computation on the same data to enable the caching of repeated accesses
  • Global data regrouping
    • Gather data used by the same computation to bring about contiguous access to memory

\[\text{a b c a a c d} \rightarrow \quad \text{a a a d d c c}\]

memory layout \[\ldots \text{a d c} \ldots\]

(a) example sequence       (b) fusing accesses on the same data       (c) grouping data used together

**Figure 1. Example use of the two-step strategy**
Global computation fusion

- Reuse-driven execution
- Firstly, collect the execution trace
- Then, reorder the instruction
- Gives priority to later instruction that Reuse the data of the current instruction

```plaintext
function Main
  for each instruction i in the ideal parallel execution order
    enqueue i to ReuseQueue
    while ReuseQueue is not empty
      dequeue i from ReuseQueue
      if (i has not been executed)
        ForceExecute(i)
    end while
  end for
end Main

function ForceExecute(instruction j)
  while there exists un-executed instruction i that produces operands for j
    ForceExecute(i)
  end while
  execute j
  for each data t used by j
    find next instruction m that uses t
    enqueue m into ReuseQueue
  end for
end ForceExecute
```
Reuse-driven execution achieves a significant reduction in number of long reuse distance

Reduce the cache miss

Figure 3. Effect of reuse-driven execution
Reuse-based loop fusion

- Because the loop contain most of data access in program
- We merge the iteration that use the same data
- Categorize data sharing into three cases
  - Loop fusion and alignment
  - Loop embedding
  - Iteration reordering

```plaintext
for i=2, N
    A[i]=f(A[i-1])
end for
A[2]=0.0

for i=3, N
    B[i]=g(A[i-2])
end for
```

```plaintext
for i=2, N
    A[i]=f(A[i-1])
    if (i==3)
        A[2]=0.0
    else if (i==N)
    end if
    if (i>2 and i<N)
        B[i+1]=g(A[i-1])
    end if
end for
B[3]=g(A[1])
```

Figure 5. Examples of reuse-based fusion
Global data regrouping

- Loop fusion does not change data layout
- So we need data regrouping to combine data used by the same loop into adjacent memory locations
- Multi-level regrouping

(a) Original program

(b) Transformed program

Figure 8. Example of multi-level regrouping
Evaluation

Table 1. Description of test programs

<table>
<thead>
<tr>
<th>name</th>
<th>source</th>
<th>input size</th>
<th>lines/loops/arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swim</td>
<td>SPEC95</td>
<td>513x513</td>
<td>429/8/15</td>
</tr>
<tr>
<td>Tomcatv</td>
<td>SPEC95</td>
<td>513x513</td>
<td>221/18/7</td>
</tr>
<tr>
<td>ADI</td>
<td>self-written</td>
<td>2Kx2K</td>
<td>108/8/3</td>
</tr>
<tr>
<td>SP</td>
<td>NAS/NPB Serial v2.3</td>
<td>class B, 3 iterations</td>
<td>1141/218/15</td>
</tr>
<tr>
<td>Sweep3D</td>
<td>DOE</td>
<td>150x150x150</td>
<td>2105/67/6</td>
</tr>
</tbody>
</table>
Figure 10. Effect of transformations
Conclusion

• Develop two global compiler strategy: reuse-based fusion loop and multi-level data regrouping to alleviate the bandwidth limitation by improving reuse of data from cache.