Hot Compilers for Future Hot Chips

Saman Amarasinghe, Jennifer Anderson, Robert French, Mary Hall, Monica Lam, Shih-Wei Liao, Brian Murphy, Chau-Wen Tseng, Chris Wilson, Robert Wilson

Stanford University

Contact: Monica Lam
(lam@cs.stanford.edu, http://sui.stanford.edu)

Microprocessor Directions

- Current trends
  - Higher clock rate
  - More instruction level parallelism (ILP)

- Alternate direction: Multiple superscalar processors on a chip
  - Hardware advantages over Very Wide superscalars:
    - Simpler, Faster
      - Less wires
      - Multiported caches → separate caches
      - Multiported registers → separate register files
      - Data dependence between instructions → disjoint sets
  - Software advantages:
    - Supports multiprogramming
    - Supports explicit parallel programming

- Key question: Performance of individual applications?
  Are multiprocessors competitive to processors with high ILP?
Potential Advantage of Multiprocessors

- High ILP $\rightarrow$ Wide variation of program performance
- Diminishing return of ILP
Challenges for Multiprocessor Compilers

- Programs with high ILP
  - Must minimize synchronization and communication cost

- Programs with low ILP
  - Multiprocessors' unique advantage →
    Extract independent threads of computation

- Programs written in C
  - Pointer analysis

SUIF Compiler Research Project

- SUIF (Stanford University Intermediate Format)

  FORTRAN / C
  \[\rightarrow\]
  SUIF
  \[\rightarrow\]
  FORTRAN / C  Code Generators

- A fully functional compiler research infrastructure
- Base version has been released and is freely available
- Techniques described will be incorporated in future releases

- Techniques expected to be transferred to industry in a few years
I. Pointer Analysis

- Important for C code
  - To reorder/parallelize loads&stores through pointers

- Efficient and accurate analysis for
  - Parameter passing
  - Pointer assignments
  - Direct array address calculations (*p++)
  - Function pointers

- Technique: Interprocedural pointer analysis algorithm
  - Avoids re-analyzing a function
    by determining the relevant pointer aliases at call sites
  - [Wilson&Lam, ACM PLDI Conference ‘95]

- Future research
  - Efficient analysis of recursive data structures

---

6.1-08

Preliminary Results

<table>
<thead>
<tr>
<th>Program</th>
<th>Lines</th>
<th>Procedures</th>
<th>Analysis Time DEC 5000 (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alvinn</td>
<td>272</td>
<td>8</td>
<td>0.29</td>
</tr>
<tr>
<td>grep</td>
<td>430</td>
<td>9</td>
<td>0.57</td>
</tr>
<tr>
<td>diff</td>
<td>668</td>
<td>23</td>
<td>1.31</td>
</tr>
<tr>
<td>lex315</td>
<td>776</td>
<td>16</td>
<td>0.85</td>
</tr>
<tr>
<td>compress</td>
<td>1503</td>
<td>14</td>
<td>1.29</td>
</tr>
<tr>
<td>loader</td>
<td>1539</td>
<td>29</td>
<td>1.43</td>
</tr>
<tr>
<td>football</td>
<td>2354</td>
<td>57</td>
<td>5.12</td>
</tr>
<tr>
<td>compiler</td>
<td>2360</td>
<td>37</td>
<td>5.86</td>
</tr>
<tr>
<td>assembler</td>
<td>3361</td>
<td>51</td>
<td>4.37</td>
</tr>
<tr>
<td>eqntott</td>
<td>3454</td>
<td>60</td>
<td>6.17</td>
</tr>
<tr>
<td>ear</td>
<td>4284</td>
<td>68</td>
<td>2.41</td>
</tr>
<tr>
<td>simulator</td>
<td>4663</td>
<td>98</td>
<td>9.89</td>
</tr>
</tbody>
</table>

- Can find all loop-level parallelism in both SPEC92fp C programs
II. Finding Coarse-Grain Parallelism

<table>
<thead>
<tr>
<th>First Generation Parallelizers</th>
<th>SUIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizations on <strong>scalar</strong> variables</td>
<td>Optimizations on <strong>array</strong> variables</td>
</tr>
<tr>
<td><strong>Within</strong> procedures</td>
<td><strong>Across</strong> procedures</td>
</tr>
</tbody>
</table>

- **Reference**
  - [Hall et al., Supercomputer 95]

---

**Reductions**

- **Generalized to reductions on (portions of) arrays**
  - in either dense or sparse representation
Privatization

- Each processor keeps a private copy of temporary data

Loop Iterations

\[
\begin{align*}
A &= &\ldots &\rightarrow & A \\
&= &A &\rightarrow & A \\
&= &A \\
A_1 &= &\ldots &\rightarrow & A_1 \\
&= &A_1 &\rightarrow & A_1 \\
&= &A_1 \\
A_2 &= &\ldots &\rightarrow & A_2 \\
&= &A_2 &\rightarrow & A_2 \\
&= &A_2 \\
A &= &\ldots &\rightarrow & A \\
&= &A &\rightarrow & A \\
&= &A \\
\end{align*}
\]

- Generalized to privatizing arrays
  - Requires data flow analysis on individual array elements

Interprocedural Analysis

- Algorithm
  - Based on interval analysis
  - Complexity: Analyze each function 3 times
  - Incremental compilation possible by saving summary information

- Important to have a complete suite of interprocedural analyses
  - Constant propagation
  - Loop invariant analysis
  - Induction variable analysis
  - Scalar privatization
  - Scalar reduction recognition
  - Array data dependence analysis
  - Array privatization
  - Reductions to regions of arrays
NAS Parallel Benchmarks on SGI Challenge

Improvement due to more parallelism:

![Graph showing speedup vs number of processors for cgms and embar(APR).]

Improvement due to less frequent synchronization:

![Graph showing speedup vs number of processors for appb and appsp.]

Observations

- **Parallelizers are maturing**
  - Interprocedural analysis
  - Array reduction and privatization

- **An interactive parallelizer will be a useful tool**
III. Optimizations

- Minimize synchronization & cache traffic across loops
  - 1st generation parallelizers
    - Insert barriers between every parallel loop
    - Distribute iterations in each loop independently
  - New global analysis
    - Smart assignment of iterations to eliminate barriers and cache traffic

**EXAMPLE: tomcatv**

```
DO J =
  DO I = (parallelizable)
  RX(I,J) = ... - RX(I,J-1)

DO L = (parallelizable)
  DO I = (parallelizable)
    ... = RX(I,L)
```

---

Data Restructuring

- **Problem**
  - False sharing: processors sharing same cache line but not data
  - Conflicts: set associativity maps data to same cache position
- **Solution:**
  - Make data accessed by each processor contiguous via data transposes and blocking
Stanford DASH Multiprocessors

- Characteristics
  - Directory-based, cache-coherent NUMA
  - 32 33MHz MIPS R3000 in 8 clusters
  - 64KB 1st level direct-mapped cache, 256KB 2nd level cache

- Techniques also useful for small-scale multiprocessors made out of fast processors

Experimental Results: Kernels

- LU 256x256
- ADI Integration 256x256
- 5 point stencil 512x512
- 1Kx1K
- 1Kx1K

Legend:
- SUIF
- 1st Generation Parallelizer
Programs

Erlebacher

Speedup

64x64x64

0 8 16 24 32
Number of Processors

Vpenta

Speedup

0 8 16 24 32
Number of Processors

SWM256

Speedup

0 8 16 24 32
Number of Processors

Tomcatv

Speedup

0 8 16 24 32
Number of Processors

---

Results on SPEC92 fp

... will be reported in the meeting
Conclusions

- **Maturation of parallelizing compilers**
  - Pointer analysis
  - Outer loop parallelism
  - Synchronization and cache optimizations
  - Interactive compilers will be effective in finding parallelism

- **Build multiprocessors with superscalar processors instead of very wide superscalars**
  - More cost-effective
  - More versatile
    - Thread and instruction level parallelism
    - Multiprogramming
  - Fine-grain parallelism possible with better support
    → multiprocessors on a chip