Optimizing Performance of Superscalar Codes for a Single Cray X1 MSP Processor

Hongzhang Shan
Erich Strohmaier

Future Technology Group
Computational Research Center
Lawrence Berkeley National Laboratory
Motivation

• Architecture
  – Merging trend of superscalar and vector.

• Application
  – Many applications developed for superscalar platform --- superacalar codes.

• Questions
  – How superscalar codes perform on new vector architectures?
  – How much programming effort needed?
- Decoupled microarchitecture
- Both Vector and Multi-stream
- Deep Memory Hierarchies: register files, cache, local memory, remote memory
Performance Implications

- Traditional performance features of vector processors:
  - Vector Length
  - Memory bank conflicts
  - Data chaining

- New features:
  - Multi-streaming
  - Memory Hierarchies

- Need to understand how performance will be affected by these factors
• Motivation
• Performance Characteristics of Cray X1
• Performance Optimization
• Summary
Apex is a project to simulate performance of applications with a synthetic benchmark.

Apex-MAP simulates application memory behavior using non-uniform random access

- $\alpha$: Data Reuse (temporal locality)
- $L$: Contiguous access length (spatial locality)
- $M$: amount of memory used

Visit http://ftg.lbl.gov
Kernel for Apex-Map

For (pos = 0; pos < LengthOfIndex; pos++)
For (i = 0; i < L; i++)
    \[\text{Sum} += \text{Data}[\text{index}\[\text{pos}\] + i]\]

Indices is generated by a power distribution function with parameter \(\alpha\)

Max Reuse: repeat same index Uniform Distribution
• Longer vector still strongly preferred.
• Vector operation for this kernel becomes efficient for $L > 16$. 

\[ M = 512\text{MB}, a = 1.0 \]
Effect of $\alpha$

L = 16, M = 512MB

- Data Reuse Matters, but effect is much smaller.
Memory bank conflicts cause significant performance loss.

For (pos = 0; pos < LengthOfIndex; pos++)
For (i = 0; i < L; i++)
    Sum += Data[index[pos] + i]

Higher Data Reuse

- Memory bank conflicts cause significant performance loss
The effect of M is also not significant.
Performance Characteristics of Cray X1

- Longer vectors most important
- Memory bank conflicts may significantly degrade the performance
- Data Reuse, memory size matters, but with relative smaller effect
- Multi-streaming is also important
• Motivation
• Performance Characteristics of Cray X1
• Performance Optimization
• Summary
<table>
<thead>
<tr>
<th>Description</th>
<th>Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conjugate Gradient Solver</td>
<td>Class C</td>
</tr>
<tr>
<td>3-D FFT</td>
<td>Class B</td>
</tr>
<tr>
<td>1-D FFT</td>
<td>16M</td>
</tr>
<tr>
<td>simulating eddy currents in an ocean basin</td>
<td>2050*2050</td>
</tr>
<tr>
<td>sorting data in ascending order using radix algorithm</td>
<td>256M</td>
</tr>
<tr>
<td>Simulation of n-body interaction in three dimensions</td>
<td>2M</td>
</tr>
<tr>
<td>dense matrix-matrix multiplication</td>
<td>2048*2048</td>
</tr>
</tbody>
</table>
Approaches

• **Compiler Directives**
  – Inside loops

• **Restructuring Application**
  – Across loops or functions
• No effect on NAS-CG, Radix
• Substantial performance improvement for other applications
  – Average 18 times better
• Compiler directives only apply to loops
• Need to eliminate data dependence between loop iterations or exploit data parallelism across loops
• Directive: vl = 16.7
• Vec-full: change *fftblock* from 16 to 256, vl=64
• No-Evolve: reduce the memory usage
• Optimized: set *fftblock* to 64, caching effect
Original:

```c
For (I = 0; I < N; I++) {
    key_val = key_from[I] & bb;
    key_val = key_val >> shift;
    bucket[key_val]++;
}
```

Optimized (Virtual Processor):

```c
For (j=0; j< VP; j++) {
    For (I=0; I < N / VP; I++) {
        key_val = key_from[j*N/VL+I] & bb;
        key_val = key_val >> shift;
        bucket[j][key_val]++;
    }
}
```
- Directive: vl = 1
- Vec-64: using 64*4 virtual processor, vl=64
- Optimized: use 63*4 virtual processors
Average Vector Length

- Average vector length becomes much longer
• Application restructuring is needed to obtain high percentage of peak performance
• Average performance for optimized kernels is 8 times better than with directives
Using *no_cache_alloc* to prevent cache line allocation for vectorized objects.

Using cache improves performance, but not so significant as restructuring, average 36% better.
MSP better for CG, Radix, otherwise, SSP better
Average MSP 11% better
Using more processors may change the results
• Superscalar codes need performance tuning on Cray X1.
• Compiler directives is the easiest and most efficient way.
• Application restructuring is important.
  – Obtaining longer vector length
  – Avoiding memory bank conflicts
  – Take cache, memory size effect into consideration
  – Multi-streaming