Computation Regrouping: Restructuring Programs for Temporal Data Cache Locality

Venkata K. Pingali
Sally A. McKee
Wilson C. Hsieh
John B. Carter

http://www.cs.utah.edu/impulse
Problem: Memory Performance

60-80% of execution time spent in memory stalls (generated by Perfex)

194 MHz, MIPS R10K Processor, 32K L1D, 32K L1I, 2MB L2
Related Work

- **Compiler approaches**
  - Loop, data and integrated restructuring: Tiling, permutation, fusion, fission [CarrMckinley94]
  - Data-centric: Multi-level fusion [DingKennedy01], Compile-time resolution [Rogers89]

- **Prefetching**
  - Hardware or software based, simple, efficient models: Jump pointers, prefetch arrays [Karlsson00], dependence-based [Roth98]

- **Cache-conscious, application-level approaches**
  - Algorithmic changes: Sorting [Lamarca96], query processing, matrix multiplication
  - Data structure modifications: Clustering, coloring, compression [Chilimbi99]
  - Application construction: Cohort Scheduling [Larus02]
Computation Regrouping

- Logical operations
  - Short streams of independent computation performing a unit task
  - Examples: R-Tree query, FFTW column walk, Processing one ray in Ray Trace

- Application-dependent optimization
  - Improve temporal locality
  - Techniques: deferred execution, early execution, filtered execution, computation merging

- Preliminary performance improvements encouraging
  - Speedups range from 1.26 to 3.03
  - Modest code changes
Access Matrix

Data Objects

Logical Operations/Time

Regrouped computations
Optimization Process Summary

- Identify data object set
  - Whose accesses result in cache misses
  - Can fit into the L2 cache

- Identify suitable computations
  - Deferrable
  - Easily parameterizable
  - Estimated gain

- Extend data/control structures
  - Extensions to store regrouped computation
  - Extensions to data structure to support partial execution

- Decide run time strategy:
  - Temporal/spatial constraints
  - Estimation of gain
Filtered Execution: IRREG

- Simplified CFD code
- Series of indirect accesses
- If index vector random, working set is as large as data array
- Memory stall accounts for more than 80% of execution time
- Logical operation: set of remote accesses

```c
for all i {
    sum += data[index[i]];
}
```

Unoptimized
Filtered Execution: IRREG

- Defer accesses to data outside the window
- Significant additional computation cost: $n$ loops instead of 1
- Tradeoff: window size vs. number of passes

```
for k = 0,n step block {
    for all i {
        if (index[i] >= k && index[i] < (k+block)){
            sum += data[index[i]];
        }
    }
}
```

Optimized
Deferred Execution: R-Tree

- Height balanced tree
- Branching factor 2-15
- Used for spatial searches
- Problem: data dependent accesses, large working set of queries/deletes
- Logical operation: insert, delete, query
R-Tree Regrouping

Query 1
Query 2
Query 3
Query 4
Regrouping: Perfex Estimates

![Bar Chart]

- **Memory**
- **TLB**
- **Computation**
- **Overhead**

<table>
<thead>
<tr>
<th>Technique</th>
<th>Memory</th>
<th>TLB</th>
<th>Computation</th>
<th>Overhead</th>
<th>Normalized Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAY TRACE</td>
<td>52</td>
<td></td>
<td></td>
<td></td>
<td>80</td>
</tr>
<tr>
<td>OPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>CUDD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>OPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>57</td>
</tr>
<tr>
<td>R-TREE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>OPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>57</td>
</tr>
<tr>
<td>EM3D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>OPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>57</td>
</tr>
<tr>
<td>IRREG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>OPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>57</td>
</tr>
</tbody>
</table>
Regrouping Vs Clustering (R-Tree)

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Cluster</th>
<th>Regrouping</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td></td>
<td>56</td>
<td>27</td>
<td>24</td>
</tr>
<tr>
<td>TLB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Computation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overhead</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Discussion

- **Downsides**
  - Useful only for a subset of inputs
  - Increased code complexity
  - Hard to automate

- **Application structure crucial to low regrouping overhead**
  - Commutative operations
  - Program-level parallelism and independence

- **Execution speed traded for output ordering and per-operation latency**
Summary

- Regrouping exploits (1) low cost of computation (2) application-level parallelism
- Improves temporal locality
- Changes small compared to overall code size
- Hand-optimized applications show good performance improvements
Implementation Techniques

- **COMPUTATION**
- **EXPENSIVE**
- **LESS EXPENSIVE**
- **NOT EXECUTED**

**ORIGINAL**

**DEFERRED EXECUTION**

**COMPUTATION MERGING**

**EARLY EXECUTION**

**FILTERED EXECUTION**

ITERATION. 1

ITERATION. 2
Deferred Execution: R-Tree

Access Matrix

Access Set Size

School of Computing

Impulse Adaptable Memory System
Performance

- SGI Power Onyx, R10K, 2MB L2, 32K L1D, 32K L1I

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Input</th>
<th>Technique</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFTW</td>
<td>10K<em>32</em>32</td>
<td>Early</td>
<td>2.53</td>
</tr>
<tr>
<td>RAY TRACE</td>
<td>Balls, 256*256</td>
<td>Filtered</td>
<td>1.98</td>
</tr>
<tr>
<td>CUDD</td>
<td>C3540.blif</td>
<td>Early + Deferred</td>
<td>1.26</td>
</tr>
<tr>
<td>IRREG</td>
<td>MOL2</td>
<td>Filtered</td>
<td>1.74</td>
</tr>
<tr>
<td>HEALTH</td>
<td>6, 500</td>
<td>Merging</td>
<td>3.03</td>
</tr>
<tr>
<td>EM3D</td>
<td>128K nodes</td>
<td>Merging + Filtered</td>
<td>1.43</td>
</tr>
<tr>
<td>R-TREE</td>
<td>dm23.in</td>
<td>Deferred</td>
<td>1.87</td>
</tr>
</tbody>
</table>
Application Analysis

- Bad memory behavior
  - Working set larger than L2
  - Data dependent accesses
  - Hard to optimize using compiler

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Source</th>
<th>Domain</th>
<th>Access Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-TREE</td>
<td>DARPA</td>
<td>Databases</td>
<td>Pointer Chasing</td>
</tr>
<tr>
<td>RAY TRACE</td>
<td>DARPA</td>
<td>Graphics</td>
<td>Pointer Chasing + Strided Accesses</td>
</tr>
<tr>
<td>CUDD</td>
<td>U. of Colorado</td>
<td>CAD</td>
<td>Pointer Chasing</td>
</tr>
<tr>
<td>EM3D</td>
<td>Public domain</td>
<td>Scientific</td>
<td>Indirect Accesses + Pointer Chasing</td>
</tr>
<tr>
<td>IRREG</td>
<td>Public Domain</td>
<td>Scientific</td>
<td>Indirect Accesses</td>
</tr>
<tr>
<td>HEALTH</td>
<td>Public Domain</td>
<td>Simulator</td>
<td>Pointer Chasing</td>
</tr>
<tr>
<td>FFTW</td>
<td>DARPA/MIT</td>
<td>Signal Processing</td>
<td>Strided Accesses</td>
</tr>
</tbody>
</table>
Thesis Overview

- Problem: complex applications increasingly limited by memory performance of applications
- Proposed approach: Computation Regrouping
- Application structure
- Generic implementation techniques
- Performance
- Simple scheduling abstraction
Characteristics of Logical Operations

- Access large number of objects
- Low reuse of data objects within a single operation
- Low computation per access
- May have high degree of reuse across operations
- Access sequence data-dependent
- Strict ordering among operations
Contributions

- Showing that computation regrouping is a viable alternative
- Characterizing the applications that can be optimized
- Developing four implementation techniques to realize computation regrouping
  - Deferred execution
  - Computation merging
  - Early execution
  - Filtered execution
- Developing simple abstraction with potential for automation (locality grouping)
Techniques Summary

- Deferred execution, e.g., R-TREE, CUDD
  - Postpone execution until sufficient computation accessing the same data is gathered

- Computation merging, e.g., HEALTH, EM3D
  - Special case of deferring
  - Application specific merging of deferred computation

- Early execution, e.g., FFTW, CUDD
  - Execute future computation that accesses the same data

- Filtered execution, e.g., IRREG, EM3D
  - Brute force technique
  - Use a sliding window to enable accesses
  - As many iterations as necessary
Deferred Execution - HEALTH

- Columbian health system simulation
- Essentially a traversal of a quad-tree and linked lists attached at nodes
- Key operation: counter update of nodes in waiting list
- Logical operation: one simulation time step
Deferred Execution - HEALTH

- Key idea: defer waiting-list traversals and remember the cumulative counter update
- Specific technique: computation merging

benefit: 1 traversal instead of many
overhead: space and processing

QUADTREE

NODE

WAITING LIST
## Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Logical Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-TREE</td>
<td>Tree operations, i.e., insert, delete and query.</td>
</tr>
<tr>
<td>RAY TRACE</td>
<td>A scan of the input scene by one ray.</td>
</tr>
<tr>
<td>CUDD</td>
<td>Hash table operations performed during variable swap.</td>
</tr>
<tr>
<td>EM3D</td>
<td>Group of accesses to a set of remote nodes.</td>
</tr>
<tr>
<td>IRREG</td>
<td>Group of accesses to a set of remote nodes.</td>
</tr>
<tr>
<td>HEALTH</td>
<td>One time step.</td>
</tr>
<tr>
<td>FFT V1</td>
<td>Column walks of a 3D array.</td>
</tr>
</tbody>
</table>
Discussion

- Correctness
  - Breaking strict logical operation ordering changes the completion and output order

- Subtle performance issues
  - Increased throughput at the cost of increased average latency, and standard deviation
  - Sensitivity to optimization parameters
R-Tree Performance Characteristics

- Synthetic input: Query operations on a large static tree

![Graph showing R-Tree performance characteristics with latency and throughput axes.](image-url)
R-Tree Performance Characteristics

![Graph showing the relationship between Queue Sizes, Average Result Latency, and Throughput. The graph highlights a 'sweet spot' where the latency is minimized.](image)

- **Average Result Latency (s)**
- **Throughput (queries/s)**

**Queue Sizes**: 0, 100, 200, 300, 400, 500, 600

**Latency (in secs)**: 0, 100, 200, 300, 400, 500, 600

**Queries/sec**: 0, 20, 40, 60, 80, 100, 120, 140
R-Tree sensitivity to Optimization Parameters

- Choice of optimization parameters is important
  - 1.4x difference between best and worst execution times
R-Tree Clustering

Intra-node Clustering

Inter-node Clustering = Intra-node Clustering +
Locality Grouping (LG)

- Locality groups: User identified groups of tasks that share objects
- Library interface
- Runtime scheduling
- Simple abstraction
  - lg *createlg(), void deletelg(lg *)
  - void addtolg(lg *, void *data, void (*proc)(void *))
  - void flushlg(lg *)
node->group = CreateLG();

if (list != NULL && only_increment){
    AddToLG(node->group, list, perform_increment);
} else {
    FlushLG(node->group);
    perform_update(list);
    ......}

DeleteLG(node->group);

AddToLG(g, arg, func){
    op = malloc();
    op.arg = arg;
    op.func = func;
    enqueue(g.ops_list, op);
}

FlushLG(g){
    while (op = dequeue()){
        (*op->func)(op->arg);
    }
}
Performance

- SGI Power Onyx, R10K, 2MB L2, 32K L1D, 32K L1I

![Performance Diagram]

- Hand-coded
- Locality Grouping

- HEALTH
- FFTW
Conclusion

- Computation regrouping is an effective software alternative
- Identified applications that can be optimized using regrouping
- Developed four implementation techniques to realize regrouping
- Demonstrated speedups ranging from 1.29 to 2.13
Acknowledgements

John, Sally, and Wilson
and
Rest of the Impulse Group
Questions ?
Key Issues

- Notion of a window
  - A sequence of computations
  - Used to capture the assumptions that the compiler can make about previous accesses
  - No 1-1 mapping between code and accesses

- Exploitation of reuse requires a large “window”
  - Many existing optimizations mostly look at a small window
  - Small window sufficient for many applications and input combinations

- Some current optimizations use large “window”
  - Multilevel-fusion [dingkennedy01]
  - Loop tiling algorithms
Regrouping Properties

- Implementation
  - Implementation supports deferring deletes and queries
  - Insert executed out-of-band
  - R-tree extensions to support correct operation

- High overhead: profitable only for large, reasonably stable trees

- Interleaving of output

- Increased throughput and operation latency
  - May be suitable for batch processing
Regrouping Example - FFTW

- Fast fourier transform implementation
- Key operation: column walks
- N column walks share the same cache lines
- Application spends between 50-90% time in these column walks
Regrouping Example - FFTW

- Fast fourier transform implementation
- Key operation: column walks
- N column walks share same cache lines
- Early execution

Overhead: control

Benefit: cache misses 1/8
Observations

- A single application may be optimized using multiple techniques

- Optimizations can be implemented to varying levels of aggressiveness

- Performance sensitive to optimization parameters
Characteristics of Logical Operations

- Access large number of objects
- Low reuse of data objects within a single operation
- May have high degree of reuse across operations
- Access sequence data-dependent
- Strict ordering among operations

Ensures that prefetching has limited impact
Characteristics of Logical Operations

- Access large number of objects
- Low reuse of data objects within a single operation
- May have high degree of reuse across operations
- Access sequence data-dependent
- Strict ordering among operations

Ensures that clustering has limited impact
Characteristics of Logical Operations

- Access large number of objects
- Low reuse of data objects within a single operation
- May have high degree of reuse across operations
- Access sequence data-dependent
- Strict ordering among operations

Ensures that large caches have limited impact