A Template Library to Integrate Thread Scheduling and Locality Management for NUMA Multiprocessors

Zoltán Majó
Thomas R. Gross

Computer Science Department
ETH Zurich, Switzerland
Non-uniform memory architecture

Local memory accesses
- bandwidth: 10.1 GB/s
- latency: 190 cycles

Remote memory accesses
- bandwidth: 6.3 GB/s
- latency: 310 cycles

Key to good performance: **data locality**

All data based on experimental evaluation of Intel Xeon 5500 (Hackenberg [MICRO ‘09], Molka [PACT ‘09])
Remote memory references (2 processors)

Remote memory references / total memory references [%]

Subset of the PARSEC benchmark suite
Outline

- Introduction
- Ferret
  - Memory behavior
  - Optimizing for data locality
  - Evaluation
- Template library
- Concluding remarks
Experimental setup

- 2-processor 8-core Intel Xeon
  - Nehalem microarchitecture
  - Linux + perfmon2
  - First-touch page placement

- Ferret: pipeline parallelism

![Diagram showing pipeline parallelism and processor setup]
Profiling memory accesses

- Data address profiling
  - Based on hardware-performance monitoring
  - Consider only heap accesses

<table>
<thead>
<tr>
<th>Profile</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 0</td>
<td>1000 accesses by Processor 0</td>
<td></td>
</tr>
<tr>
<td>Page 1</td>
<td>3000 accesses by Processor 1</td>
<td></td>
</tr>
</tbody>
</table>

Diagram showing memory accesses between Processor 0 and Processor 1, with Page 0 and Page 1 for each processor.
Profiling memory accesses

- Data address profiling
  - Based on hardware-performance monitoring
  - Consider only heap accesses

<table>
<thead>
<tr>
<th>Profile</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 0</td>
<td>1000 accesses by Processor 0</td>
</tr>
<tr>
<td>Page 1</td>
<td>3000 accesses by Processor 1</td>
</tr>
</tbody>
</table>
| Page 2  | 4000 accesses by Processor 0  
|         | 5000 accesses by Processor 1 |

Ferret: **41%** of profiled pages inter-processor shared
Detailed look

- **Ferret: similarity search of images**
  - Input: set of images
  - Output: list of images similar to input

- **Index stage: most memory system intensive**
  - Produces 90% of total memory bandwidth

![Diagram of image search process]

Image database


Input → Segment → Extract → Index → Rank → Output

Candidate set

Query
Locality-sensitive hashing

<table>
<thead>
<tr>
<th>Hash value</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>...</th>
<th>H-1</th>
<th>H</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Image ID</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
</tr>
<tr>
<td>6</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N-1</td>
<td>...</td>
</tr>
<tr>
<td>N</td>
<td>...</td>
</tr>
</tbody>
</table>
Locality-sensitive hashing

Database System

Index

Data index

<table>
<thead>
<tr>
<th>Hash value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>H-1</td>
</tr>
<tr>
<td>H</td>
</tr>
</tbody>
</table>

Image data

<table>
<thead>
<tr>
<th>Image ID</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
</tr>
<tr>
<td>6</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>N-1</td>
<td>...</td>
</tr>
<tr>
<td>N</td>
<td>...</td>
</tr>
</tbody>
</table>

Index

query

query
Bad data locality

- Data accesses and parallelization *decoupled*
  - Database interface hides details about internal structure
  - All threads running the indexing phase access database

- Optimizing for data locality
  - Precise control of data and computation allocation *within database*
  - Example: 2-processor system
Optimizing for data locality

<table>
<thead>
<tr>
<th>Data Index</th>
<th>Image data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash value</td>
<td>Image ID</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
</tr>
<tr>
<td>6</td>
<td>N-1</td>
</tr>
<tr>
<td>...</td>
<td>N</td>
</tr>
<tr>
<td>(H-1)</td>
<td></td>
</tr>
<tr>
<td>(H)</td>
<td></td>
</tr>
</tbody>
</table>
Optimizing for data locality

### Database System

<table>
<thead>
<tr>
<th>Thread pool</th>
<th>Data Index</th>
<th>Image data: CPU 0</th>
<th>Image data: CPU 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU 0</strong></td>
<td>Hash value</td>
<td>Image ID</td>
<td>Features</td>
</tr>
<tr>
<td>T0 T0</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>T0 T0</td>
<td>2</td>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>H-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **CPU 1**   |          |          |            |          |            |
| T1 T1       |          |          |            |          |            |
| T1 T1       |          |          |            |          |            |

---

**Thread pool**

- **CPU 0**: T0 T0, T0 T0
- **CPU 1**: T1 T1, T1 T1
Optimizing for data locality

Database System

Thread pool

CPU 0
T0 T0
T0 T0

CPU 1
T1 T1
T1 T1

Data Index

Hash value
1
2
3
4
5
6
H-1
H

Image data: CPU 0

Image ID Features
1 ...
2 ...
3 ...
N/2 ...

Image data: CPU 1

Image ID Features
N/2 + 1 ...
N/2 ...
N-1 ...
N ...
Evaluation

- **Two machines**
  - 2-processor **8-core** Intel Nehalem (Xeon E5520)
  - 4-processor **32-core** Intel Westmere (Xeon E7-4830)

- **Two image database sizes**
  - **small**: ~60 K images
  - **large**: ~744 K images
  - 3500 image queries in both cases

- **Compare two configurations**
  - **default**: first-touch page allocation, affinity scheduling
  - **NUMA-aware** memory allocation and computation scheduling
Data locality

Remote memory references / total memory references [%]

8-core machine

32-core machine

- **default**
- **NUMA-aware**

---

small | large
---|---

8-core machine

small | large
---|---

32-core machine
Performance

Performance improvement of NUMA-aware over default [%]

8-core machine

32-core machine

small

large

small

large
Optimizing for data locality – summary

- Optimizing data locality in ferret difficult
  - System developer: internals of shared database must be understood
  - Database developer: system must be understood
- Data structures often key to good NUMA performance
- Template library: basis for NUMA-aware data structures
Template library

- **Base class**
  - Per-processor data allocation
  - Locality-aware task dispatch

```csharp
SplittableData<Data, Result>
Data newAtProcessor(p: Processor)
Result dispatch(queryTask: Task)
```
abstract class SplittableData<Data,Result> {
    ThreadPool threadPool;
    Map<Processor,Data> map;

    Data newAtProcessor(Processor p) {
        // new Data instance at Processor p
        // record mapping Processor → Data
    }

    Result dispatch(Task queryTask) {
        List<Result> results;
        Data localData;
        for (Processor p : processors) {
            localData = map.get(p);
            threadPool.submit(queryTask, p, localData);
        }
        for (Processor p : processors)
            results.add(threadPool.get(p));
        return Result.merge(results);
    }
}
Concluding remarks

- Some parallel programs NUMA-unfriendly *by construction*
- Good performance: *programmer intervention required*
- Template library to abstract low-level system details
Thank you for your attention!