Data-Flow Awareness in Parallel Data Processing

D. Bednárek, J. Dokulil*, J. Yaghob, F. Zavoral

Charles University Prague, Czech Republic

*University of Vienna, Austria
Problem

- Processing large data collections (GB, TB)
  - unstructured data, BigData
  - data streams

- Cloud Computing
  - Hadoop
  - Map-Reduce paradigm
  - not sufficient for nonlinear computations
    - matrix inversion, fluid dynamics equations, data joins
Problem

- Parallel processing
  - Threading Building Blocks
    - shared memory parallel processing
  - OpenMP
    - mathematical computations
  - MPI
    - distributed computing
- Significant knowledge required
  - synchronization, cache hierarchy, technical details
Bobox

- Framework for parallel programming
- Main goal: simplify writing parallel programs
- Targeted to data-intensive applications
- Technical details handled by the framework
Data flow awareness

- Data-flow awareness in parallel data processing

- Explicit specification of data flow
  - part of the application algorithm
  - gives the task scheduler more precise info
  - improves the quality of task scheduling

⇒ better performance
Scheduler

- Scheduler - selection of the next task
- TBB, OpenMP, or MPI
  - no information on data flow
  - difficult to optimize their strategy
- Bobox model
  - abstraction of data flow known to the scheduler
    - decisions are data-flow aware
  - box - basic computational component
    - zero or more inputs and outputs in the pipeline
  - via - one link in the pipeline
    - connects one output to one or more inputs of other boxes
  - envelope - the smallest unit of data
Scheduling in Bobox

- Fixed number of threads
- Separate scheduler for each thread
  - one central task scheduler would create a bottleneck
  - immediate queue
    - tasks to be scheduled immediately
    - data are hot in cache
  - stealing queue
    - tasks which are not tightly bound to the thread
- Scheduling algorithm
  - execute the first task in the immediate queue
  - if empty - head of the stealing queue
  - if empty - steals tasks from another scheduler
Task enqueueing

- Task enqueueing
  - immediate task - head of the immediate queue
  - relaxed task - end of the stealing queue
  - hard-wired into a box and via framework

- Main reason: cache-awareness of the framework
  - head of the immediate queue should be scheduled immediately by the same thread on the same CPU
  - the data bound to the task is hot in the cache
  - number of memory accesses is minimized

- relaxed task is placed on the tail
  - since the data are not needed to be hot
Scheduling example

(a) Start

(b) Init finished

(c) Boxes scheduled

(d) Task stolen

(e) Boxes finished

(f) Another task stolen
1. start

- Bobox model - pipeline
- 4 threads, task queues
- initial box is enqueued
2. initialization finished

- box produces data
- sends it to the first via
- enqueues the via
3. boxes scheduled

- the via duplicates data
- sends it to the boxes
- enqueues them
4. task stolen

- thread 2 steals one task
- both tasks are invoked
- boxes produce their results
5. boxes finished

- newly created tasks
- enqueued with the same thread
6. another task stolen

- one task stolen by the thread 3
Performance

- DFA and non-DFA (referential) scheduler
- Performance is affected by envelope (data) size
  - data larger than cache $\Rightarrow$ performance decreases
  - several levels of caches $\Rightarrow$ several drops
- Number of threads
  - CPU: 4 cores w/ hyperthreading
  - 2 to 4 threads significantly better than single-threaded
  - best results: 7 threads
Performance / envelope size

![Graph showing performance over envelope size. The x-axis represents envelope size in KB (2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, 512K). The y-axis represents time in milliseconds (ms) ranging from 0 to 4500. The graph compares 'flow aware' and 'flow unaware' scenarios. The 'flow aware' scenario shows significantly lower time compared to 'flow unaware' for all envelope sizes. There's a notable increase in time for 'flow unaware' as the envelope size increases, whereas 'flow aware' remains relatively constant.](image-url)
Performance / number of threads

The graph shows the time (in milliseconds) on the y-axis and the number of threads on the x-axis. The line represents the time taken for the system to perform a task as the number of threads increases. The dotted line indicates the standard deviation. There are three green arrows pointing to specific points on the graph, which might indicate areas of interest or significance in the performance metrics.

Key observations:
- The time decreases significantly as the number of threads increases from 1 to 4.
- There is a slight increase in time around 7 threads.
- The time remains relatively stable after 10 threads.

The graph suggests an optimal number of threads where the performance is maximized.
Related projects

- Bobox is used in several related projects
  - SPARQL compiler
  - model visualization (graph drawing)
  - semantic processing
  - query optimization
  - data stream processing (astro-informatics)
- Participating universities
  - Charles University Prague (cz)
  - Comenius University Bratislava (sk)
  - University of Vienna (at)
Conclusions

- We have demonstrated the impact of supplying dataflow information to a scheduler
- Advantage of Bobox architecture
  - data-flow information is a part of the application code
- Data-awareness
  - significant performance improvement
  - DFA scheduler required appx. 30% less time
Thank you for your attention
Bobox runtime architecture
## Performance / envelope size

<table>
<thead>
<tr>
<th></th>
<th>2K</th>
<th>4K</th>
<th>8K</th>
<th>16K</th>
<th>32K</th>
<th>64K</th>
<th>128K</th>
<th>256K</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFA</td>
<td>1686</td>
<td>1267</td>
<td>1046</td>
<td>947</td>
<td>911</td>
<td>977</td>
<td>1420</td>
<td>2935</td>
</tr>
<tr>
<td>ref</td>
<td>1927</td>
<td>1423</td>
<td>1313</td>
<td>1297</td>
<td>1751</td>
<td>3002</td>
<td>3007</td>
<td>4044</td>
</tr>
</tbody>
</table>
Box

- computation unit
- basic paradigms
  - task parallelism, non-linear pipeline
- high-performance messaging
  - communication, synchronization
- technical details handled by the framework
  - NUMA, cache hierarchy, CPU architecture