Experiments in Multicore and Distributed Processing Using JCSP

Jon Kerridge
School of Computing
Edinburgh Napier University
Introduction

• Scottish Informatics and Computer Science Alliance issued a multi-core challenge:
  – To evaluate the effectiveness of parallelising applications to run on multi-core processors initially using a Concordance example.

• Additionally, an MSc student hand undertaken experiments using a Monte Carlo $\pi$ algorithm with multi-threaded solutions in a .NET environment, which had given some surprising results.

• Repeated the student experiments using JCSP to see what differences, if any, from the .NET results
Software Environment

- **Groovy**
  - A Java based scripting language
    - Direct support for Lists and Maps
  - Executes on a standard JVM
- **JCSP**
  - A CSP based library for Java
  - Process definitions independent of how the system will be executed
  - Enables multicore parallelism
  - Parallelism over a distributed system with TCP/IP interconnect
  - Executes on a standard JVM
- A set of Groovy Helper Classes have been created to permit easier access to the JCSP library
Student Experience - Saeed Dickie

• Showed, in .NET framework that if you added many threads then the overall processing time increased.

• The multi-core processor tended to spend most of its time swapping between threads.

• The CPU usage was 100%, but did not do useful work

• This could be observed using the Visual Studio 2010 Concurrency Visualizer
Monte Carlo pi

• If a circle of radius $R$ is inscribed inside a square with side length $2R$, then the area of the circle will be $\pi R^2$ and the area of the square will be $(2R)^2$. So the ratio of the area of the circle to the area of the square will be $\pi /4$.

• So select a large number of points at random
• Determine whether the point is within or out with the inscribed circle
• Calculate the ratio
Monte Carlo pi - Parallelisation

- Split the iterations over a number of workers
- Each will calculate its own count of the number of points within circle
- Combine all the values to get the overall count to calculate pi
- The more workers the faster the solution should appear
# Machines Used

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>cores</th>
<th>speed</th>
<th>L2 cache</th>
<th>RAM GB</th>
<th>OS</th>
<th>Size bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Office</td>
<td>E8400</td>
<td>2</td>
<td>3.0</td>
<td>6</td>
<td>2</td>
<td>XP</td>
<td>32</td>
</tr>
<tr>
<td>Home</td>
<td>Q8400</td>
<td>4</td>
<td>2.66</td>
<td>4</td>
<td>8</td>
<td>Windows 7</td>
<td>64</td>
</tr>
<tr>
<td>Lab</td>
<td>E8400</td>
<td>2</td>
<td>3.0</td>
<td>8</td>
<td>2</td>
<td>Windows 7</td>
<td>32</td>
</tr>
</tbody>
</table>
## Single Machine

<table>
<thead>
<tr>
<th></th>
<th>Office (secs)</th>
<th>Home (secs)</th>
<th>Lab (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>4.378</td>
<td>2.448</td>
<td>4.508</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Workers</th>
<th>Speedup</th>
<th>Speedup</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4.621</td>
<td>0.947</td>
<td>2.429</td>
</tr>
<tr>
<td>4</td>
<td>4.677</td>
<td>0.936</td>
<td>8.171</td>
</tr>
<tr>
<td>8</td>
<td>4.591</td>
<td>0.954</td>
<td>7.827</td>
</tr>
<tr>
<td>16</td>
<td>4.735</td>
<td>0.925</td>
<td>7.702</td>
</tr>
<tr>
<td>32</td>
<td>4.841</td>
<td>0.904</td>
<td>7.601</td>
</tr>
<tr>
<td>64</td>
<td>4.936</td>
<td>0.887</td>
<td>7.635</td>
</tr>
<tr>
<td>128</td>
<td>5.063</td>
<td>0.865</td>
<td>7.541</td>
</tr>
</tbody>
</table>
Conclusion – Not Good

• Apart from the Home Quad Core Machine with 2 workers all the other options showed a slow-down rather than a speed up
• The slow-down got worse as the number of parallel increased
• The Java JVM plus Windows OS is not able to allocate parallels over the cores effectively

• So
  • How about running each worker in a separate JVM?
  • Would each JVM be executed in a separate core?

• It is crucial to note that the Worker and Manager processes have not changed; just the manner of their invocation.
## Outcome

<table>
<thead>
<tr>
<th>JVMs</th>
<th>Office Time (secs)</th>
<th>Office Speed up</th>
<th>Home Time (secs)</th>
<th>Home Speed up</th>
<th>Lab Time (secs)</th>
<th>Lab Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4.517</td>
<td>0.969</td>
<td>2</td>
<td>2.195</td>
<td>1.115</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4.534</td>
<td>0.966</td>
<td>4</td>
<td>1.299</td>
<td>1.885</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4.501</td>
<td>0.973</td>
<td>8</td>
<td>1.362</td>
<td>1.797</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some Improvement

• The Windows 7 machines, Home and Lab showed speedups
• The XP machine did not, even though it is the same specification as the Lab machine

• So what happens if we run the system on multiple machines

• The processes and manner of invocation do not need to be changed
• Just run them on separate machines.
• They interact with a separate process called the NodeServer that organises the actual network channels
• This could only be run on Lab type machines
## Distributed Multi JVM operation

<table>
<thead>
<tr>
<th>Two Machines</th>
<th>JVMs</th>
<th>Time (secs)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab</td>
<td>2</td>
<td>4.371</td>
<td>1.031</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2.206</td>
<td>2.044</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Four Machines</th>
<th>JVMs</th>
<th>Time (secs)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab</td>
<td>4</td>
<td>2.162</td>
<td>2.085</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1.229</td>
<td>3.668</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>1.415</td>
<td>3.186</td>
</tr>
</tbody>
</table>

There are only 8 cores available on 4 machines
Montecarlo Conclusions

• Run each worker in its own JVM
• Only use the same number of workers as there are cores
• Speedup will be compatible with the number of machines
• Use an environment where it is easy to place processes on machines
  – Design the system parallel from the outset
• Distribute the application over machines
  – Then use the extra cores

• The original goal of Intel in designing multi-core processors was to reduce heat generation.
  – They did not expect all cores to be used simultaneously.
  – They expected cores to be used for applications not processes
The SICSA Concordance Challenge

- **Given:** Text file containing English text in ASCII encoding. An integer N.
- **Find:** For all sequences of words, up to length N, occurring in the input file, the number of occurrences of this sequence in the text, together with a list of start indices. Optionally, sequences with only 1 occurrence should be omitted.
Concordance

• Essentially this is an I/O bound problem and thus not easy to parallelise
• The challenge thus is to extract parallelism wherever possible
• The largest text available was the bible comprising
  – Input file 4.6MB
  – Output file 25.8MB
    • N = 6; At least two occurrence of each word string
  – 802,000 words in total

• The Lab Machine environment was used
  – A network of dual core machines
Design Decisions

• Use many distributed machines
• Do not rely on the individual cores
• Ensure all data structures are separable in some parameter
  – N in this case
  – Reduces contention for memory access;
  – Hence easier to parallelise
• Keep loops simple
  – Easier to parallelise
Architecture

There can be any number of workers; in these experiments 4, 8 and 12 Bi-directional CSP channel communication in Client-Server Design
Read File process

• Reads parameters
  – input file name, N value, Minimum number of repetitions to be output
  – Number of workers and Block size

• Operation
  – Reads input file, tokenises into space delimited words
  – Forms a block of such words ensuring an overlap of N-1 words between blocks
  – Sends a block to each worker in turn

  – Merges the final partial concordance of each worker and writes final concordance to an output file
    • Will be removed in the final version
Initial Experiments

• The relationship between Block Size and the Number of Workers governs how much processing can be overlapped with the initial file input

• It was discovered that for Block Size = 6144 gave the best performance for 4 or 8 workers

• Provided the only work undertaken was
  – removal of punctuation and
  – the initial calculation of the equivalent integer value for each word
Worker – Initial Phase

- Reads input blocks from Read File process
  - Removes punctuation – saving as bare words
  - Calculates integer equivalent value for each word by summing its ASCII characters
    - This is also the $N = 1$ sequence value
  - These operations are overlapped with input and the same process in each worker

- For each block
  - Calculate the integer value for each sequence of length 2 up to $N$ by adding word values and store it in a Sequence list

- The integer values generated by this processing will generate duplicate values for different words and different sequences
Worker – Local Map Generation

• For each Sequence in each Block
  – Produce a Map of the Sequence value with the corresponding entry of a Map comprising the corresponding word strings with an entry of the places where that word string is found in the input file
  – Save this in a structure that is indexed by N and each contains a list of the Maps produced above

• For each worker produce a composite Map combining the individual Maps
  – Save this in a structure indexed by N
  – This is the Concordance for this worker
Worker – Merge Phase

- For each of the N partial Concordances
  - Sort the integer keys into descending order
  - For each Key in the Nth partial Concordance
    - Send the corresponding Map Entry to the Reader
    - The Map Entry contains a Map of the word sequences and locations within file

- This will be modified in the final version that overlaps the merge / output phase
Worker - Parallelisation

- Each Worker can be parallelised by \( N \)
- Data structures indexed by \( N \) can be written to in parallel
  - Provided each element of the parallel only accesses a single value of \( N \)
  - Access to any shared structures is read only

- Thus depending on the number of available machines these operations can be carried out in parallel

- Thus the design is scalable in \( N \) and machines
## Equal Speedup Analysis

<table>
<thead>
<tr>
<th>Worker Style</th>
<th>Workers</th>
<th>Time (secs)</th>
<th>Speedup by workers</th>
<th>Speedup by style</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>138</td>
<td>1.99</td>
<td>2.58</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>70</td>
<td>1.99</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>54</td>
<td>1.94</td>
<td>2.58</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>28</td>
<td>1.94</td>
<td>2.52</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>18</td>
<td>2.98</td>
<td></td>
</tr>
</tbody>
</table>
Commentary - Overall

Merge Effects

• For N = 3
  – The Merge time is very similar
  – Demonstrates that the Merge is the bottleneck

<table>
<thead>
<tr>
<th>Worker Total Time (secs)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>W = 4</td>
<td>1.40</td>
</tr>
<tr>
<td>W = 8</td>
<td>1.73</td>
</tr>
<tr>
<td>W = 8</td>
<td>1.24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>Total Time (secs)</th>
<th>Time Ratio</th>
<th>Output File Size MB</th>
<th>Size Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>44</td>
<td>1.40</td>
<td>18</td>
<td>1.20</td>
</tr>
<tr>
<td>4</td>
<td>62</td>
<td>1.41</td>
<td>21</td>
<td>1.20</td>
</tr>
<tr>
<td>5</td>
<td>82</td>
<td>1.86</td>
<td>24</td>
<td>1.34</td>
</tr>
<tr>
<td>6</td>
<td>102</td>
<td>2.34</td>
<td>26</td>
<td>1.45</td>
</tr>
</tbody>
</table>

Time ratio much greater than size ratio

Merge Parallelisation

• There is an option here to parallelise more by undertaking merges in parallel
Overlapped Merge / Output Architecture

Reader

Worker

Worker

Merge N = 1

Merge N = 2

Merge N = 3
Commentary on Revised Architecture

• The workers output each of the N Primary maps in parallel to the respective Merge process
  – Each worker has N processes that output the entries in each primary key map in descending sorted order
  – One merge process per N value
  – Each Merge process writes its own file

• When the worker has finished
  – Sends a message to Reader informing it of termination
  – This enables calculation of overall time

• The architecture implements the CSP Client-Server design pattern thereby guaranteeing freedom from deadlock
## Worker Style Time Ratios W=12

<table>
<thead>
<tr>
<th>N</th>
<th>Total Time (secs)</th>
<th>Time Ratio</th>
<th>Output File Size MB</th>
<th>Size Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>44</td>
<td>1.36</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>62</td>
<td>1.41</td>
<td>21</td>
<td>1.20</td>
</tr>
<tr>
<td>5</td>
<td>82</td>
<td>1.86</td>
<td>24</td>
<td>1.34</td>
</tr>
<tr>
<td>6</td>
<td>103</td>
<td>2.34</td>
<td>26</td>
<td>1.45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Worker Style</th>
<th>N</th>
<th>Total Time (secs)</th>
<th>Time Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq</td>
<td>3</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>seq</td>
<td>6</td>
<td>103</td>
<td>1.61</td>
</tr>
<tr>
<td>par</td>
<td>3</td>
<td>32</td>
<td>1.36</td>
</tr>
<tr>
<td>par</td>
<td>6</td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>
## Ratio Analysis for Different Sources

<table>
<thead>
<tr>
<th>12 Workers</th>
<th>Words</th>
<th>Total Output MB</th>
<th>Output for N = 1 KB</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bible</td>
<td>802,300</td>
<td>26</td>
<td>6,297</td>
<td>64</td>
</tr>
<tr>
<td>WaD</td>
<td>268,500</td>
<td>5.4</td>
<td>2,044</td>
<td>27</td>
</tr>
<tr>
<td>Ratio</td>
<td>2.99</td>
<td>4.76</td>
<td>3.08</td>
<td>2.34</td>
</tr>
</tbody>
</table>

WaD – Wives and Daughters
Conclusion

• Utilisation of access to shared memory needs to be considered when designing the algorithm
  – This was done from the outset with the choice of data structures

• The parallelisation of sequential sections is relatively straightforward
  – Provided there are no memory access violations between parallel processes
  – The JCSP Library made this particularly easy

• The resulting system is scalable in
  – The number of Workers
  – The value of N and the number of available machines
  – 19 machines used in this implementation
Real Conclusion

More Questions than Answers