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 π 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

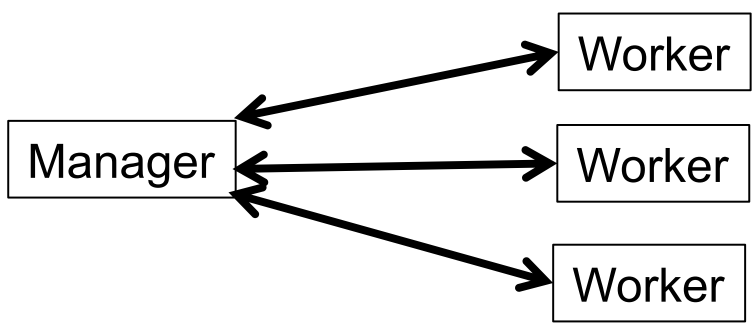
Monte Carlo pi

- If a circle of radius R is inscribed inside a square with side length 2R,
- then the area of the circle will be πR^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 outwith 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

	CPU	cores	speed Ghz	L2 cache MB	RAM GB	OS	Size bits
Office	E8400	2	3.0	6	2	XP	32
Home	Q8400	4	2.66	4	8	Windows 7	64
Lab	E8400	2	3.0	8	2	Windows 7	32

Single Machine

	Workers	Office (secs)	Speedup	Home (secs)	Speedup	Lab (secs)	Speedup
Sequential		4.378		2.448		4.508	
Parallel	2	4.621	0.947	2.429	1.008	4.724	0.954
	4	4.677	0.936	8.171	0.300	4.685	0.962
	8	4.591	0.954	7.827	0.313	4.902	0.920
	16	4.735	0.925	7.702	0.318	4.897	0.921
	32	4.841	0.904	7.601	0.322	5.022	0.898
	64	4.936	0.887	7.635	0.321	5.161	0.873
	128	5.063	0.865	7.541	0.325	5.319	0.848

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

	Office			Home			Lab	
JVMs	Time (secs)	Speed up	JVMs	Time (secs)	Speed up	JVMs	Time (secs)	Speed up
2	4.517	0.969	2	2.195	1.115	2	4.369	1.032
4	4.534	0.966	4	1.299	1.885	4	4.323	1.043
8	4.501	0.973	8	1.362	1.797	8	4.326	1.042

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

Two Machines	JVMs	Time (secs)	Speedup
Lab	2	4.371	1.031
	4	2.206	2.044

Four Machines	JVMs	Time (secs)	Speedup
Lab	4	2.162	2.085
	8	1.229	3.668
	16	1.415	3.186

There are only 8 cores available on 4 machines

Edinburgh Napier UNIVERSITY

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

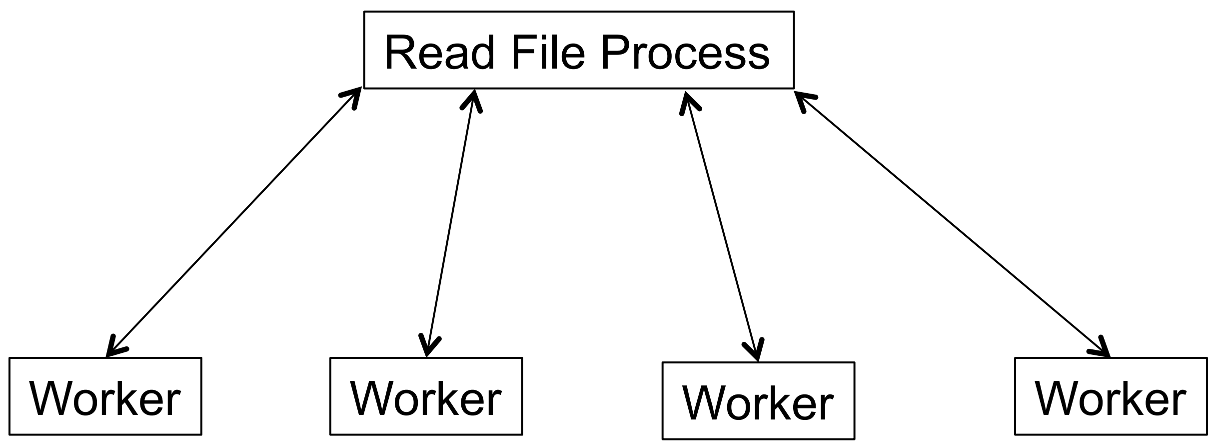
- Essentially this is an I/O bound problem and thus not easy to parallelise
- The challenge thus is to extract parallelism wherever possible
- The largest text available was the bible comprising
  - Input file 4.6MB
  - Output file 25.8MB for
    - N = 6; At least two occurrence of each word string
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

Edinburgh Napier UNIVERSITY

Worker Style	Workers	Time (secs)	Speedup by workers	Speedup by style
1	4	138		
1	8	70	1.99	
2	4	54		2.58
2	8	28	1.94	2.52
2	12	18	2.98	

Commentary - Overall

Edinburgh Napier UNIVERSITY

Merge Effects

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

Worker Total Time Speedup

	W = 8	W = 12
W = 4	1.40	1.73
W = 8		1.24

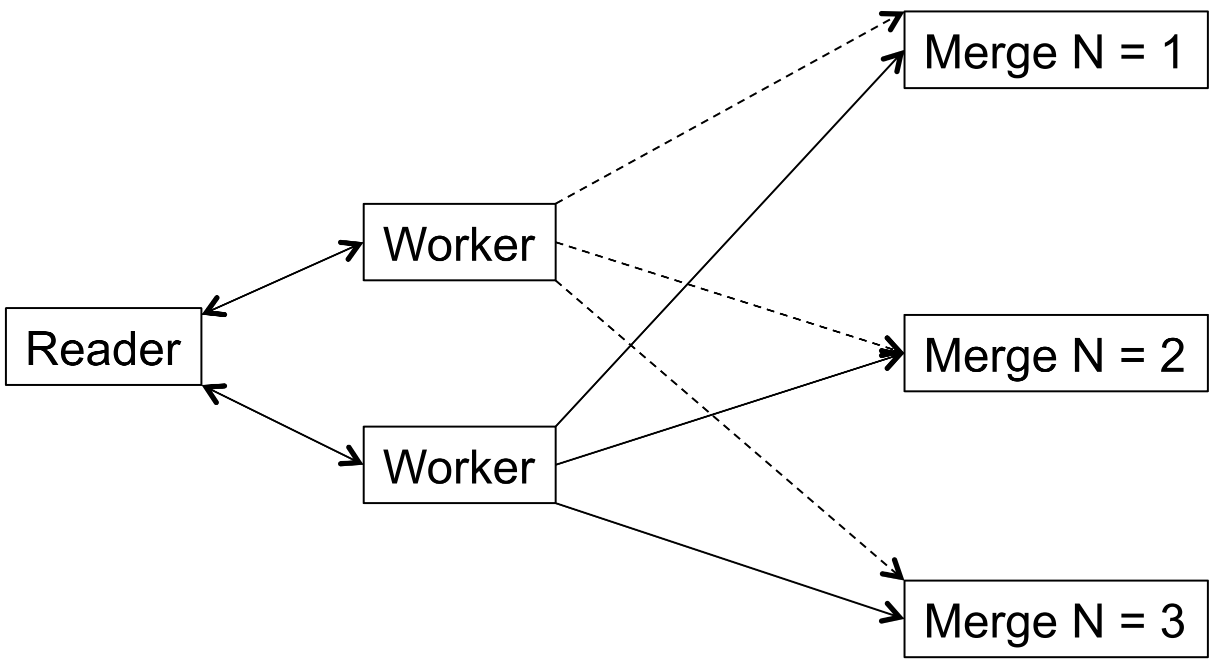
N	Total Time (secs)	Time Ratio	Output File Size MB	Size Ratio
3	44		18	
4	62	1.41	21	1.20
5	82	1.86	24	1.34
6	102	2.34	26	1.45

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



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

Edinburgh Napier UNIVERSITY

Worker Style Time Ratios W=12

N	Total Time (secs)	Time Ratio	Output File Size MB	Size Ratio
3	44		18	
4	62	1.41	21	1.20
5	82	1.86	24	1.34
6	103	2.34	26	1.45

Worker Style	N	Total Time (secs)	Time Ratio
seq	3	44	
seq	6	103	
par	3	32	1.36
par	6	64	1.61

Ratio Analysis for Different Sources

12 Workers	Words	Total Output MB	Output for N = 1 KB	Time (secs)
Bible	802,300	26	6,297	64
WaD	268,500	5.4	2,044	27
Ratio	2.99	4.76	3.08	2.34

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