Optimization Space
Exploration of the FastFlow Parallel Skeleton Framework

Alexander Collins      Christian Fensch      Hugh Leather
University of Edinburgh
School of Informatics
What We Did

- Parallel skeletons provide easy abstraction for parallel programs
- Contain many manually tuned parameters
- **Automatically tuning provides better performance**
- Preliminary results that make auto-tuning faster
Motivating Example

What we want to compute $\rightarrow$ $\text{fib}(6)$
Motivating Example

What we want to compute

fib(6)

fib(4)  fib(5)

Divide phase
Motivating Example

What we want to compute

fib(6) - Divide phase

fib(4) - Stop recursion at fib(3) or less

fib(2)

fib(3)

fib(5)

fib(3)

fib(4)

fib(2)

fib(3)
Motivating Example

What we want to compute

fib(6)

fib(4)  fib(5)

fib(2)  fib(3)  fib(3)  fib(4)

Stop recursion at fib(3) or less

1  2  2  1  2

Divide phase
Sequential base cases
Motivating Example

What we want to compute

Stop recursion at fib(3) or less

Divide phase
Sequential base cases
Conquer phase
Motivating Example

What we want to compute

Stop recursion at fib(3) or less

Final result

1
3
5
8
1
2
3
2
1
2

 divides phase
Sequential base cases
Conquer phase
Tuning the Example

![Graph showing speedup vs. maximum recursion depth]

- Speedup on the y-axis.
- Maximum recursion depth on the x-axis.
- Points indicating human expert and best possible speedup.

Key:
- Human expert
- Best possible
What Next?

- Humans failed
- Auto-tuning won
- Can we do even better?
  - Multiple parameters
  - Multiple programs
  - Multiple platforms
Optimization Space Exploration

Parameter values: 1 7 3 3 9

Program -> Compiler -> Executable

Monte Carlo Search

Multi-core Machine

Execution Times

Results
Parameters Investigated

- Number of workers
- Bounded or unbounded queues
- Size of queue’s buffer
- Cache alignment
- Maximum recursion depth
- Batch size
Speedup over a Human Expert

Program: dt

Speedup

Platform

- 2-cores
- 4-cores
- 6-cores
- 16-cores
- 32-cores
- Average
Speedup over a Human Expert

Platform: 6-cores

<table>
<thead>
<tr>
<th>Program</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>aquad</td>
<td>2.2</td>
</tr>
<tr>
<td>cwc</td>
<td>1.5</td>
</tr>
<tr>
<td>dt</td>
<td>1.7</td>
</tr>
<tr>
<td>fib.</td>
<td>0.8</td>
</tr>
<tr>
<td>mbrot</td>
<td>1.2</td>
</tr>
<tr>
<td>matmul</td>
<td>1.1</td>
</tr>
<tr>
<td>nqueens</td>
<td>1.0</td>
</tr>
<tr>
<td>pbzip2</td>
<td>1.0</td>
</tr>
<tr>
<td>qsort</td>
<td>1.0</td>
</tr>
<tr>
<td>swps3</td>
<td>1.0</td>
</tr>
<tr>
<td>Average</td>
<td>1.5</td>
</tr>
</tbody>
</table>
Speedup over a Human Expert

![Bar chart showing speedup over a human expert for different platform sizes. The chart is labeled "All Programs" and shows speedup on a scale from 0 to 3 on the y-axis. The platforms are 2-cores, 4-cores, 6-cores, 16-cores, 32-cores, and Average. The 32-cores platform has the highest speedup, followed by the 16-cores platform. The speedup for the 2-cores platform is the lowest.]
Visualising the Optimisation Space
**Visualisation of Optimisation Space**

<table>
<thead>
<tr>
<th>Program</th>
<th>2 core</th>
<th>4 core</th>
<th>6 core</th>
<th>16 core</th>
<th>32 core</th>
</tr>
</thead>
<tbody>
<tr>
<td>cwc</td>
<td><img src="image1" alt="Heatmap" /></td>
<td><img src="image2" alt="Heatmap" /></td>
<td><img src="image3" alt="Heatmap" /></td>
<td><img src="image4" alt="Heatmap" /></td>
<td><img src="image5" alt="Heatmap" /></td>
</tr>
<tr>
<td>dt</td>
<td><img src="image6" alt="Heatmap" /></td>
<td><img src="image7" alt="Heatmap" /></td>
<td><img src="image8" alt="Heatmap" /></td>
<td><img src="image9" alt="Heatmap" /></td>
<td><img src="image10" alt="Heatmap" /></td>
</tr>
<tr>
<td>pbzip2</td>
<td><img src="image11" alt="Heatmap" /></td>
<td><img src="image12" alt="Heatmap" /></td>
<td><img src="image13" alt="Heatmap" /></td>
<td><img src="image14" alt="Heatmap" /></td>
<td><img src="image15" alt="Heatmap" /></td>
</tr>
<tr>
<td>swps3</td>
<td><img src="image16" alt="Heatmap" /></td>
<td><img src="image17" alt="Heatmap" /></td>
<td><img src="image18" alt="Heatmap" /></td>
<td><img src="image19" alt="Heatmap" /></td>
<td><img src="image20" alt="Heatmap" /></td>
</tr>
</tbody>
</table>
Reducing the Size of the Search Space

- Two methods:
  - Remove useless parameters
  - Exploit linear dependencies
What is a Useless Parameter?

Small area = Small effect on performance
What is a Useful Parameter?

![Diagram showing the relationship between parameter value and performance. The area under the curve represents the effect on performance. The text on the image says: Large area = Large effect on performance.]
Removing Useless Parameters

- Reduces size of search space by $6 \times$
Exploiting Linear Dependencies
Exploiting Linear Dependencies
Exploiting Linear Dependencies

![Bar Chart]

- Variability of Data Captured
- Number of Parameters Used

<table>
<thead>
<tr>
<th>Parameters Used</th>
<th>Variability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20%</td>
</tr>
<tr>
<td>2</td>
<td>30%</td>
</tr>
<tr>
<td>3</td>
<td>40%</td>
</tr>
<tr>
<td>4</td>
<td>50%</td>
</tr>
<tr>
<td>5</td>
<td>70%</td>
</tr>
<tr>
<td>6</td>
<td>80%</td>
</tr>
</tbody>
</table>
Conclusions

- Tuning parameters is very important
- Humans are bad at tuning
- **Auto-tuning is much better**
- Tuning is program and platform dependent
- Have shown preliminary results that make auto-tuning faster
Optimization Space
Exploration of the FastFlow Parallel Skeleton Framework

Alexander Collins    Christian Fensch    Hugh Leather

University of Edinburgh
School of Informatics
Speedup over a Human Expert

- aquad
- cwc
- dt
- fibonacci
- mandelbrot
- matmul
- nqueens
- pbzip2
- quicksort
- swps3

Average

Desktop
Phantom
Scuttle
xxxii16
xxxii
Average
Principal Components Analysis

\(N = 6 \quad P = 5624\)

\(p = \begin{pmatrix}
\text{batchsize, buffersize, buffertype,}
\text{cachealign, numworkers, seqthresh}
\end{pmatrix}\)

\(\lambda = (0.443, 0.419, 0.204, 0.138, 0.007, 0.003)\)

\(e = \begin{pmatrix}
0.023 & 0.814 & -0.000 & -0.003 & -0.580 & 0.013 \\
-0.015 & 0.581 & 0.002 & 0.002 & 0.814 & -0.014 \\
-0.115 & -0.001 & 0.005 & -0.000 & 0.015 & 0.993 \\
0.993 & -0.010 & -0.000 & -0.001 & 0.028 & 0.115 \\
-0.001 & -0.001 & 0.003 & -1.000 & 0.003 & -0.001 \\
-0.001 & 0.001 & -1.000 & -0.003 & 0.002 & 0.005
\end{pmatrix}\)

\(\nu = (36\%, 70\%, 87\%, 99\%, 99\%, 100\%)\)
Is the subset representative?
<table>
<thead>
<tr>
<th>Program</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aquad</td>
<td>Adaptive Quadrature algorithm</td>
</tr>
<tr>
<td>cwc</td>
<td>Implementation of CWC, a calculus for the representation and simulation of biological systems</td>
</tr>
<tr>
<td>dt</td>
<td>Implementation of the C4.5 decision tree algorithm</td>
</tr>
<tr>
<td>fibonacci</td>
<td>Naïve recursive algorithm, without memoization, to compute Fibonacci numbers</td>
</tr>
<tr>
<td>mandelbrot</td>
<td>Mandelbrot fractal generator</td>
</tr>
<tr>
<td>matmul</td>
<td>$O(n^3)$ nested-loops matrix multiplication</td>
</tr>
<tr>
<td>nqueens</td>
<td>$n$-queens problem solver</td>
</tr>
<tr>
<td>pbzip2</td>
<td>Parallel bzip2 compression</td>
</tr>
<tr>
<td>quicksort</td>
<td>Parallel quicksort</td>
</tr>
<tr>
<td>swps3</td>
<td>Smith-Waterman algorithm for gene sequence alignment</td>
</tr>
</tbody>
</table>
## Platforms

<table>
<thead>
<tr>
<th>Platform</th>
<th>Processor</th>
<th>Cores</th>
<th>Freq.</th>
<th>Memory</th>
<th>L3</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxii</td>
<td>4× Intel Xeon L7555</td>
<td>32</td>
<td>1.87GHz</td>
<td>64GB</td>
<td>4× 24MB</td>
<td>32× 256KB</td>
</tr>
<tr>
<td>xxxii16</td>
<td>2× Intel Xeon L7555</td>
<td>16</td>
<td>1.87GHz</td>
<td>64GB</td>
<td>2× 24MB</td>
<td>16× 256KB</td>
</tr>
<tr>
<td>scuttle</td>
<td>AMD Phenom II X6 1055T</td>
<td>6</td>
<td>3.3GHz</td>
<td>8GB</td>
<td>1× 6MB</td>
<td>1× 512KB</td>
</tr>
<tr>
<td>phantom</td>
<td>Intel Xeon E5430</td>
<td>4</td>
<td>2.67GHz</td>
<td>8GB</td>
<td>None</td>
<td>2× 6MB</td>
</tr>
<tr>
<td>desktop</td>
<td>Intel Core 2 Duo E6400</td>
<td>2</td>
<td>2.13GHz</td>
<td>3GB</td>
<td>None</td>
<td>1× 2MB</td>
</tr>
</tbody>
</table>
### Parameter Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>numworkers</td>
<td>$1, \ldots, #\text{ cores} \times 1.5$</td>
</tr>
<tr>
<td>buffertype</td>
<td>Bounded or unbounded</td>
</tr>
<tr>
<td>buffersize</td>
<td>$1, 2, 4, 8, \ldots, 2^{20}$</td>
</tr>
<tr>
<td>batchsize</td>
<td>$1, 2, 4, 8, \ldots, 2^{20}$</td>
</tr>
<tr>
<td>cachealign</td>
<td>64, 128 or 256 bytes</td>
</tr>
<tr>
<td>seqthresh with aquad</td>
<td>0.02, 0.04, 0.06, \ldots, 1</td>
</tr>
<tr>
<td>seqthresh with fibonacci</td>
<td>10, 11, 12, \ldots, 44</td>
</tr>
<tr>
<td>seqthresh with nqueens</td>
<td>3, 4, 5, \ldots, 15</td>
</tr>
<tr>
<td>seqthresh with quicksort</td>
<td>$1, 2, 4, 8, \ldots, 2^{21}$</td>
</tr>
</tbody>
</table>
Outlier Removal

- Arithmetic mean is not a robust statistic
- An outlier will cause many more repeats
- Impractical
- Remove using interquartile range removal:
  \[
  [Q_1 - k(Q_3 - Q_1), Q_3 + k(Q_3 - Q_1)]
  \]
  with \( k = 3 \)
Quantifying Noise

- Repeats allow quantification of noise:
  - Perform between 10 and 100 repeats
  - Stop if coefficient of variation drops below 1% for a 99% confidence interval
- Use the arithmetic mean as an estimator of execution time
- And confidence intervals to compare execution times
Skeletons Provided by FastFlow

- **farm**
  - Emitter
  - Worker 1
  - Worker n
  - Collector

- **farm-with-feedback**
  - Emitter
  - Worker 1
  - Worker n
  - Collector

- **pipe**
  - Stage 1
  - …
  - Stage n