Auto-Mobiles
Optimising Message-Passing Concurrency

Neil Brown

School of Computing
University of Kent
UK

2 November 2009
Auto-Mobiles

- Compiler Optimisation
  - For message-passing languages like occam 2
  - Increases speed
  - Reduces memory use
- No change to your code required
Best Case

![Graph showing time in microseconds against array size in bytes for different array sizes: 2, 8, and 64.](image-url)
Best Case

Time (microseconds) vs Array Size (bytes)

- 2
- 8
- 64
Auto-Mobiles

- Compiler Optimisation
  - For message-passing languages like occam 2
  - Increases speed
  - Reduces memory use
- No change to your code required
Data Operations
Write
Read
Communication
Reading
Writing
Communication
Problems with occam 2
Statically Sized Storage
Communication
Slow for Large Data
Slow for Large Data
Slow for Large Data
Mobility
occam 2 + faster 

mobiles \downarrow 

occam-π ←
Reading
Writing
Communication
Read
Write
Communication
Communication
Communication
Nothing Gained (Yet)
Communication
Communication
Communication
Communication
Communication
Idea behind Auto-Mobiles

- Make all non-tiny data mobile (i.e. a heap object)
- Normally, copy and send reference
- If you don’t need your copy afterwards, send your original
- The recipient doesn’t need to know which you did!
Idea behind Auto-Mobiles

- Make all non-tiny data mobile (i.e. a heap object)
- Normally, copy and send reference
  - If you don’t need your copy afterwards, send your original
  - The recipient doesn’t need to know which you did!
Idea behind Auto-Mobiles

- Make all non-tiny data mobile (i.e. a heap object)
- Normally, copy and send reference
- If you don’t need your copy afterwards, send your original
- The recipient doesn’t need to know which you did!
Idea behind Auto-Mobiles

- Make all non-tiny data mobile (i.e. a heap object)
- Normally, copy and send reference
- If you don’t need your copy afterwards, send your original
- The recipient doesn’t need to know which you did!
How do we know if we still need it?
How do we know if we still need it?

Compile-Time Program Flow Analysis
PROC id (CHAN FOO in?, out!)
    FOO x:
    WHILE TRUE
        SEQ
            in ? x
            out ! x
    :
Flow Analysis Examples

PROC id (CHAN FOO in?, out!)
  FOO x:
  WHILE TRUE
    SEQ
      in ? x
      out ! x
    :
PROC id.100 (CHAN FOO in?, out!)

    FOO x:
    SEQ i = 0 FOR 100
    SEQ
        in ? x
        out ! x
    :
Flow Analysis Examples

PROC id.100 (CHAN FOO in?, out!)
    FOO x:
    SEQ i = 0 FOR 100
    SEQ
        in ? x
        out ! x
    :
Simplifying assumptions:

- All data same size
- Consistent communication behaviour
- Allocation takes negligible time compared to copying
Speedup Bounds

Simplifying assumptions:

- All data same size
- Consistent communication behaviour
- Allocation takes negligible time compared to copying

Speedup Factor Bound \( = \frac{1}{1-M} \)
Benchmark 1 of 3: The Ring
Benchmark 1 of 3: The Ring

![Graph showing time (microseconds) vs. array size (bytes) for different configurations of The Ring benchmark.](image)
Benchmark 1 of 3: The Ring

![Graphs showing time (microseconds) vs. array size (bytes) for different thread counts. The graph on the left shows a trend where time increases with array size, while the graph on the right shows a flat line, indicating constant time regardless of array size.]

University of Kent Computing
Benchmark 2 of 3: The Twin Pipeline
Benchmark 2 of 3: The Twin Pipeline
Benchmark 2 of 3: The Twin Pipeline
Benchmark 2 of 3: The Twin Pipeline

![Graph showing benchmark results]

- **Plain time divided by mobile time**
- **Array Size (bytes)**
  - 64
  - 256
  - 1KB
  - 4KB
  - 16KB
  - 64KB
  - 256KB
  - 1MB
  - 4MB
  - 16MB

Legend:
- 1
- 8
- 64
Benchmark 3 of 3: occam audio kit (oak)
Summary

\[ \texttt{occam}^2 + \texttt{mobiles} \rightarrow \texttt{faster} \]

\[ \Downarrow \]

\[ \texttt{occam-\pi} \]
Summary
Communication
Summary

Communication
PROC id (CHAN FOO in?, out!)
  FOO x:
  WHILE TRUE
  SEQ
    in ? x
    out ! x
: 
Conclusions

- It’s already in Tock, enabled by compiler flag
- Better speed and memory use than plain occam 2
- No extra programmer burden (no change to code)
- Retains simple copy semantics
- Opportunity cost of mobile data ideas (e.g. mobile atoms)
  - Can use mobile channels as tokens
- Supporting dynamically-sized arrays is now easy
Allocation vs Copying

\[ c(S) > (1 - M)(c(S) + a(S)) \]

\[ \frac{M}{1-M} > \frac{a(S)}{c(S)} \]
Copy-on-Write