Buffer Sharing in CSP-like Programs

Nalini Vasudevan    Stephen A. Edwards

Columbia University
Motivation

Task 1

\[ a = 6; \]
\[ \text{send } a; \]

Task 2

\[ \text{recv } a; \]
\[ b = a + 1; \]
\[ \text{send } b; \]

Task 3

\[ \text{recv } b; \]
\[ c = b \times 2; \]
\[ \text{send } c; \]

Task 4

\[ \text{recv } c; \]
Motivation

- Use rendezvous model of communication

Task 1

\[
\begin{align*}
a &= 6; \\
send a;
\end{align*}
\]

Task 2

\[
\begin{align*}
recv a; \\
b &= a + 1; \\
send b;
\end{align*}
\]

Task 3

\[
\begin{align*}
recv b; \\
c &= b * 2; \\
send c;
\end{align*}
\]

Task 4

\[
recv c;
\]
Motivation

Task 1

\[ a = 6; \]
\[ send a; \]

Task 2

\[ recv a; \]
\[ b = a + 1; \]
\[ send b; \]

Task 3

\[ recv b; \]
\[ c = b \times 2; \]
\[ send c; \]

Task 4

\[ recv c; \]
Motivation

Task 1

\[ a = 6; \]
\[ \text{send } a; \]

Task 2

\[ \text{recv } a; \]
\[ b = a + 1; \]
\[ \text{send } b; \]

Task 3

\[ \text{recv } b; \]
\[ c = b \times 2; \]
\[ \text{send } c; \]

Task 4

\[ \text{recv } c; \]

Shared Memory

write

Sender

read

Receiver

Buffer Sharing in CSP-like Programs, Nalini Vasudevan and Stephen A. Edwards, Columbia University – p. 3
Motivation

Task 1

\[ a = 6; \]
\[ \text{send } a; \]

Task 2

\[ \text{recv } a; \]
\[ b = a + 1; \]
\[ \text{send } b; \]

Task 3

\[ \text{recv } b; \]
\[ c = b \times 2; \]
\[ \text{send } c; \]

Task 4

\[ \text{recv } c; \]

Shared Memory

write

Sender

read

Receiver

- a, b and c can share buffers
Buffer Sharing

Task 1

\[ a = 6; \]
\[ \text{send } a; \]

Task 2

\[ \text{recv } a; \]
\[ b = a + 1; \]
\[ \text{send } b; \]

Task 3

\[ \text{recv } b; \]
\[ c = b \times 2; \]
\[ \text{send } c; \]

Task 4

\[ \text{recv } c; \]
Buffer Sharing

Task 1

\[ a = 6; \]
\[ \text{send } a; \]

Task 2

\[ \text{recv } a; \]
\[ b = a + 1; \]
\[ \text{send } b; \]

Task 3

\[ \text{recv } b; \]
\[ c = b \times 2; \]
\[ \text{send } c; \]

Task 4

\[ \text{recv } c; \]
Buffer Sharing

Task 1

\[ a = 6; \]
\[ \text{send } a; \]

Task 2

\[ \text{recv } a; \]
\[ b = a + 1; \]
\[ \text{send } b; \]

Task 3

\[ \text{recv } b; \]
\[ c = b \times 2; \]
\[ \text{send } c; \]

Task 4

\[ \text{recv } c; \]
Buffer Sharing

Task 1

\[a = 6;\]
\[\text{send } a;\]

Task 2

\[\text{recv } a;\]
\[b = a + 1;\]
\[\text{send } b;\]

Task 3

\[\text{recv } b;\]
\[c = b \times 2;\]
\[\text{send } c;\]

Task 4

\[\text{recv } c;\]
Buffer Sharing

Task 1

\[ a = 6; \]
\[ \text{send } a; \]

Task 2

\[ \text{recv } a; \]
\[ b = a + 1; \]
\[ \text{send } b; \]

Task 3

\[ \text{recv } b; \]
\[ c = b \times 2; \]
\[ \text{send } c; \]

Task 4

\[ \text{recv } c; \]
Automata Composition

Task 1

1

2

a

Task 2

1

2

a

b

Task 3

1

2

b

c

Task 4

1

2

c
Automata Composition
Another Example

Task 1

\[
\text{for } (\text{int } i = 0; i < 15; i++)
\]

\[
\text{recv } a;
\]

\[
\text{send } b = 10;
\]

\[
\text{send } d = 8;
\]

Task 2

\[
\text{recv } b;
\]

\[
\text{recv } c;
\]
Another Example

Task 1

\[
\text{for (int } i = 0; i < 15; i++)
\]

\[
\text{recv a;}
\]

\[
\text{send } b = 10;
\]

\[
\text{send } d = 8;
\]

Task 2

\[
\text{recv } b;
\]

\[
\text{recv } c;
\]
Another Example

Task 1

```
for (int i = 0; i < 15; i++)
    recv a;

send b = 10;
send d = 8;
```

Task 2

```
recv b;
recv c;
```

- a, b and d can share buffers
- b and c can share buffers
- a and c can share buffers
- c and d cannot share buffers
Another Example

Task 1

\[
\text{for (} \text{int } i = 0; i < 15; i++) \\
\text{recv } a; \\
\text{send } b = 10; \\
\text{send } d = 8;
\]

Task 2

\[
\text{recv } b; \\
\text{recv } c;
\]
Another Example

Task 1

\begin{verbatim}
for (int i = 0; i < 15; i++)
    recv a;
send b = 10;
send d = 8;
\end{verbatim}

Task 2

recv b;
recv c;

Buffer Sharing in CSP-like Programs, Nalini Vasudevan and Stephen A. Edwards, Columbia University – p. 8
Automata Composition

Task 1

1

2

3

a

b

d

Task 2

1

2

3

b

c

d

Buffer Sharing in CSP-like Programs, Nalini Vasudevan and Stephen A. Edwards, Columbia University – p. 9
Automata Composition

Task 1

1
2
3

Task 2

1
2
3

Buffer Sharing in CSP-like Programs, Nalini Vasudevan and Stephen A. Edwards, Columbia University – p. 9
Automata Composition

- c and d cannot share buffers
- False positive: a and b cannot share buffers
Grouping Channels

- a and b can share buffers
- b and c can share buffers
- a and c cannot share buffers

Two possibilities
- \{a,b\} \{c\}
- \{b,c\} \{a\}
Grouping Channels

- a and b can share buffers
- b and c can share buffers
- a and c cannot share buffers

Two possibilities

- \{a,b\} \{c\}
- \{b,c\} \{a\}

Suppose

- a: 2MB
- b: 8MB
- c: 8MB
Grouping Channels

Greedy, first-fit method

- b: 8MB
- c: 8MB
- a: 2MB
Grouping Channels

Greedy, first-fit method

- b: 8MB
- c: 8MB
- a: 2MB

{b}
Grouping Channels

Greedy, first-fit method

- b: 8MB
- c: 8MB
- a: 2MB

{b}
{b,c}
Grouping Channels

Greedy, first-fit method

- b: 8MB
- c: 8MB
- a: 2MB

\{b\}
\{b,c\}
\{b,c\} \{a\}
## Results

<table>
<thead>
<tr>
<th>Example</th>
<th>Lines</th>
<th>Channels</th>
<th>Tasks</th>
<th>Bytes Saved</th>
<th>Buffer Reduction</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source-Sink</td>
<td>35</td>
<td>2</td>
<td>11</td>
<td>4</td>
<td>50 %</td>
<td>0.1 s</td>
</tr>
<tr>
<td>Pipeline</td>
<td>35</td>
<td>5</td>
<td>9</td>
<td>16388</td>
<td>25</td>
<td>0.1</td>
</tr>
<tr>
<td>Bitonic Sort</td>
<td>35</td>
<td>5</td>
<td>13</td>
<td>12</td>
<td>60</td>
<td>0.1</td>
</tr>
<tr>
<td>Prime Sieve</td>
<td>40</td>
<td>5</td>
<td>16</td>
<td>12</td>
<td>60</td>
<td>0.5</td>
</tr>
<tr>
<td>Berkeley</td>
<td>40</td>
<td>3</td>
<td>11</td>
<td>4</td>
<td>33.33</td>
<td>0.6</td>
</tr>
<tr>
<td>FIR Filter</td>
<td>110</td>
<td>28</td>
<td>28</td>
<td>52</td>
<td>46.43</td>
<td>13.8</td>
</tr>
<tr>
<td>Framebuffer</td>
<td>185</td>
<td>11</td>
<td>16</td>
<td>28</td>
<td>0.002</td>
<td>1.3</td>
</tr>
<tr>
<td>FFT</td>
<td>230</td>
<td>14</td>
<td>15</td>
<td>344068</td>
<td>50</td>
<td>0.6</td>
</tr>
<tr>
<td>JPEG Decoder</td>
<td>1020</td>
<td>7</td>
<td>15</td>
<td>772</td>
<td>50.13</td>
<td>1.8</td>
</tr>
</tbody>
</table>
Related Work

- Significant work in sequential programs
  [Greef et al., ASAP ’97]
- Synchronous data flow
  [Murthy et al., ACM TODAES ’04]
- Constrain the schedule to save memory
  [Chrobak et al., ICALP ’01]
Conclusions

- Reduces memory without affecting the run-time schedule
- Can be applied to the Cell compiler
  - Can save 344 kB of PPE’s memory for FFT
- Future work
  - More modular techniques
  - Reduce memory in k-place buffered models