Look Into Details: The Benefits of Fine-Grain Streaming Buffer Analysis

Mohammad H. Foroozannejad
Matin Hashemi
Trevor L. Hodges
Soheil Ghiasi

University of California, Davis, CA
Streaming Applications

- Widespread
  - Cell phones, mp3 players, video conference, real-time encryption, graphics, HDTV editing hyperspectral imaging, cellular base stations

- Definition
  - Infinite sequence of data items
  - At any given time, operates on a small window of this sequence
  - Moves forward in data space

```c
// 53° around the z axis
const R[3][3] = {
    {0.6, -0.8, 0.0},
    {0.8, 0.6, 0.0},
    {0.0, 0.0, 1.0}
};

Rotation3D {
    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
            B[i] += R[i][j] * A[j]
}
```
Application Model

- **Data Flow Graph**
  - Vertices or Actors
    - functions, computations
  - Edges
    - data dependency, communication between actors
- **Execution Model**
  - any actor can perform its computation whenever all necessary input data are available on incoming edges.
An example Data Flow Graph: Vocoder

http://www.cag.csail.mit.edu/streamit
**SDF (Synchronous Data Flow Graph)** is one special case
- Fixed input and output rates on the edges
- statically schedulable
Software Synthesis from SDF

while(1)
    for i = 0..1
        for j = 0..2
            X[i*3+j] = S[j] + 1
        for j = 3..6
            Y[i*4+j] = S[j] ^ 2
    for i = 0..5
        Z[i] = X[i] ^ 3
    for i = 0..3
        T[i] = (Y[i*2]+Y[i*2+1])^0.5
    P = 0
    for i = 0..5
        P = Z[i] + P
        if (P >= 0)
            Out(P+T[0]+T[2])
        else
            Out(-P+T[1]+T[3])
end While
Software Synthesis from SDF

while(1)
    for i = 0..1
        for j = 0..2
            X[i*3+j] = S[j] + 1
        for j = 3..6
            Y[i*4+j] = S[j] ^ 2
    for i = 0..3
        Z[i] = X[i] ^ 3
        T[i] = (Y[i*2] + Y[i*2+1]) ^ 0.5
        P = 0
        for i = 0..5
            P = Z[i] + P
            if (P >= 0)
                Out(P+T[0]+T[2])
            else
                Out(-P+T[1]+T[3])
    end While

Schedule:  2A  6B  4C  D
Firing Sequence:  A  A  B  B  B  B  B  C  C  C  C  D

Diagram:

SB

YT

XZ
Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B C C C C D

while(1)
for i = 0..1
  for j = 0..2
    X[i*3+j] = S[j] + 1
  for j = 3..6
    Y[i*4+j] = S[j] ^ 2
for i = 0..5
  Z[i] = X[i] ^ 3
for i = 0..3
  T[i] = (Y[i*2]+Y[i*2+1])^0.5
P = 0
for i = 0..5
  P = Z[i] + P
  if (P >= 0)
    Out(P+T[0]+T[2])
  else
    Out(-P+T[1]+T[3])
end While
Software Synthesis from SDF

while(1)
    for i = 0..1
        for j = 0..2
            X[i*3+j] = S[j] + 1
        for j = 3..6
            Y[i*4+j] = S[j] ^ 2
    for i = 0..5
        Z[i] = X[i] ^ 3
    for i = 0..3
        T[i] = (Y[i*2]+Y[i*2+1])^0.5

    P = 0
    for i = 0..5
        P = Z[i] + P
    if (P >= 0)
        Out(P+T[0]+T[2])
    else
        Out(-P+T[1]+T[3])
end While
Software Synthesis from SDF

while(1)
    for i = 0..1
        for j = 0..2
            X[i*3+j] = S[j] + 1
        for j = 3..6
            Y[i*4+j] = S[j] ± 2
    for i = 0..5
        Z[i] = X[i] ± 3
    for i = 0..3
        T[i] = (Y[i*2]+Y[i*2+1]) ± 0.5
    P = 0
    for i = 0..5
        P = Z[i] + P
        if (P >= 0)
            Out(P+T[0]+T[2])
        else
            Out(-P+T[1]+T[3])
end While

Schedule: 2A 6B 4C D
Firing Sequence: A A B B B B B C C C C D
Software Synthesis from SDF

```plaintext
while(1)
  for i = 0..1
    for j = 0..2
      X[i*3+j] = S[j] + 1
      for j = 3..6
        Y[i*4+j] = S[j] ^ 2
    for i = 0..5
      Z[i] = X[i] ^ 3
  for i = 0..3
    T[i] = (Y[i*2]+Y[i*2+1])^0.5

P = 0
for i = 0..5
  P = Z[i] + P
if (P >= 0)
  Out(P+T[0]+T[2])
else
  Out(-P+T[1]+T[3])
end While
```

Schedule: 2A 6B 4C D
Firing Sequence: A A B B B B B C C C C D
Software Synthesis from SDF

while(1)
  for i = 0..1
    for j = 0..2
      X[i*3+j] = S[j] + 1
    for j = 3..6
      Y[i*4+j] = S[j] ^ 2
  for i = 0..5
    Z[i] = X[i] ^ 3
  for i = 0..3
    T[i]=(Y[i*2]+Y[i*2+1])^0.5

P = 0
for i = 0..5
  P = Z[i] + P
  if (P >= 0)
    Out(P+T[0]+T[2])
  else
    Out(-P+T[1]+T[3])
end While
Software Synthesis from SDF

while(1)
    for i = 0..1
        for j = 0..2
            X[i*3+j] = S[j] + 1
        for j = 3..6
            Y[i*4+j] = S[j] ^ 2
    for i = 0..5
        Z[i] = X[i] ^ 3
    for i = 0..3
        T[i] = (Y[i*2]+Y[i*2+1])^0.5
    P = 0
    for i = 0..5
        P = Z[i] + P
    if (P >= 0)
        Out(P+T[0]+T[2])
    else
        Out(-P+T[1]+T[3])
end While

Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B C C C C D
Software Synthesis from SDF

```
while(1)
  for i = 0..1
    for j = 0..2
      X[i*3+j] = S[j] + 1
    for j = 3..6
      Y[i*4+j] = S[j] ^ 2
  for i = 0..5
    Z[i] = X[i] ^ 3
  for i = 0..3
    T[i] = (Y[i*2]+Y[i+2])^0.5
  P = 0
  for i = 0..5
    P = Z[i] + P
    if (P >= 0)
      Out(P+T[0]+T[2])
    else
      Out(-P+T[1]+T[3])
end While
```
Software Synthesis from SDF

while(1)
    for i = 0..1
        for j = 0..2
            X[i*3+j] = S[j] + 1
        for j = 3..6
            Y[i*4+j] = S[j] ^ 2
    for i = 0..5
        Z[i] = X[i] ^ 3
    for i = 0..3
        T[i]=(Y[i*2]+Y[i*2+1])^0.5
end While

P = 0
for i = 0..5
    P = Z[i] + P
    if (P >= 0)
        Out(P+T[0]+T[2])
    else
        Out(-P+T[1]+T[3])
end for

Schedule: 2A 6B 4C D
Firing Sequence: A A B B B B B C C C C D
Software Synthesis from SDF

while (1)
    for i = 0..1
        for j = 0..2
            X[i*3+j] = S[j] + 1
        for j = 3..6
            Y[i*4+j] = S[j] ^ 2
    for i = 0..3
        Z[i] = X[i] ^ 3
    for i = 0..3
        T[i] = (Y[i*2]+Y[i*2+1]) ^ 0.5
    P = 0
    for i = 0..5
        P = Z[i] + P
        if (P >= 0)
            Out(P+T[0]+T[2])
        else
            Out(-P+T[1]+T[3])
end While

Schedule: 2A 6B 4C D
Firing Sequence: A A B B B B B C C C C D
Software Synthesis from SDF

while(1)
    for i = 0..1
        for j = 0..2
            X[i*3+j] = S[j] + 1
        for j = 3..6
            Y[i*4+j] = S[j] ^ 2
    for i = 0..5
        Z[i] = X[i] ^ 3
    for i = 0..3
        T[i] = (Y[i*2]+Y[i*2+1]) ^ 0.5

    P = 0
    for i = 0..5
        P = Z[i] + P
    if (P >= 0)
        Out(P+T[0]+T[2])
    else
        Out(-P+T[1]+T[3])
end While

Schedule: 2A 6B 4C D
Firing Sequence: A A B B B B B C C C C D
Software Synthesis from SDF

while(1)
  for i = 0..1
    for j = 0..2
      X[i*3+j] = S[j] + 1
    for j = 3..6
      Y[i*4+j] = S[j] ^ 2
  for i = 0..5
    Z[i] = X[i] ^ 3
  for i = 0..3
    T[i] = (Y[i*2] + Y[i*2+1]) ^ 0.5

P = 0
for i = 0..5
  P = Z[i] + P
if (P >= 0)
  Out(P + T[0] + T[2])
else
  Out(-P + T[1] + T[3])
end While

Schedule:  2A  6B  4C  D
Firing Sequence:  A  A  B  B  B  B  B  C  C  C  C  D
Software Synthesis from SDF

while(1)
    for i = 0..1
        for j = 0..2
            X[i*3+j] = S[j] + 1
        for j = 3..6
            Y[i*4+j] = S[j] ^ 2
    for i = 0..5
        Z[i] = X[i] ^ 3
    for i = 0..3
        T[i] = (Y[i*2]+Y[i*2+1])^0.5

P = 0
for i = 0..5
    P = Z[i] + P
if (P >= 0)
    Out(P+T[0]+T[2])
else
    Out(-P+T[1]+T[3])
end While

Schedule:      2A    6B    4C    D
Firing Sequence:        A   A   B   B   B   B   B   B   C   C   C   C   D
Shared Buffer Implementation

- **Idea:**
  - Most of the time channel buffers are completely or partially empty.

- **Rules:**
  1. No over-writing or reading another buffer’s data.
  2. Statically allocated
  3. No re-allocation
Visualizing Buffer Analysis

- Tow dimensional plane
  - X-axis: Actor firings in the schedule (time)
  - Y-axis: Buffer location in the memory (space)
  - Filled Area: The range between Head and Tail indices

- Advantage:
  - Memory allocation problem can be viewed as a geometric layout instance
  - A solution is valid when the laid out buffers do not conflict in the time-memory plane.
Visualizing Buffer Analysis

Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B C C C C C D

for i = 1..2
X[i] = S[i] + 1
for i = 3..7
Y[j] = S[i] * 2

Z[i] = X[i] ^ 3
P = 0
for i = 0..5
P = Z[i] + P
if (P >= 0)
Out(P+T[0]+T[2])
else
Out(-P+T[1]+T[3])

T[i] = (Y[i*2]+Y[i*2+1]) ^ 0.5

Index in Memory

Actor Firings
Visualizing Buffer Analysis

For $i = 1..2$
$X[i] = S[i] + 1$

For $i = 3..7$
$Y[j] = S[i] ^ 2$

$Z[i] = X[i] ^ 3$

$P = 0$
For $i = 0..5$
$P = Z[i] + P$
If $(P >= 0)$
Out $(P + T[0] + T[2])$
Else
Out $(-P + T[1] + T[3])$

$T[i] = (Y[i*2] + Y[i*2+1]) ^ 0.5$

Schedule: 2A 6B 4C D
Firing Sequence: A A B B B B B C C C C C D
Visualizing Buffer Analysis

Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B C C C C D

\[
\begin{align*}
& \text{for } i = 1..2 \quad X[i] = S[i] + 1 \\
& \text{for } i = 3..7 \quad Y[j] = S[i] ^ 2 \\
& \text{for } i = 0..5 \\
& P = Z[i] + P \quad \text{if } (P \geq 0) \\
& \text{Out}(P + T[0] + T[2]) \quad \text{else} \\
& \text{Out}(-P + T[1] + T[3]) \\
& T[i]=(Y[i*2]+Y[i*2+1])^{0.5}
\end{align*}
\]
Visualizing Buffer Analysis

Index in Memory

S

A

B

C

D

\[ P = 0 \]
\[ \text{for } i = 0..5 \]
\[ P = Z[i] + P \]
\[ \text{if (} P \geq 0) \]
\[ \text{Out}(P+T[0]+T[2]) \]
\[ \text{else} \]
\[ \text{Out}(-P+T[1]+T[3]) \]

\[ T[i] = (Y[i*2]+Y[i*2+1])^{0.5} \]

Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B C C C C D
Visualizing Buffer Analysis

Schedule: 2A 6B 4C D
Firing Sequence: A A B B B B C C C C D

\[ Z[i] = X[i] ^ 3 \]

\[ T[i] = (Y[i*2] + Y[i*2+1]) ^ 0.5 \]

\[ P = 0 \]
\[ \text{for } i = 0..5 \]
\[ P = Z[i] + P \]
\[ \text{if } (P \geq 0) \]
\[ \text{Out}(P+T[0]+T[2]) \]
\[ \text{else} \]
\[ \text{Out}(-P+T[1]+T[3]) \]
Visualizing Buffer Analysis

Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

\[
\begin{align*}
X[i] &= S[i] + 1 \\
Y[j] &= S[i] ^ 2 \\
Z[i] &= X[i] ^ 3 \\
T[i] &= (Y[i+2]+Y[i+2+1]) ^ 0.5 \\
P &= 0 \\
\text{for } i = 0..5 \\
P &= Z[i] + P \\
\text{if } (P \geq 0) \\
\text{Out}(P+T[0]+T[2]) \\
\text{else} \\
\text{Out}(-P+T[1]+T[3])
\end{align*}
\]
Visualizing Buffer Analysis

For $i = 1..2$
\[ X[i] = S[i] + 1 \]
For $i = 3..7$
\[ Y[j] = S[i] ^ 2 \]

\[ Z[i] = X[i] ^ 3 \]
\[ P = 0 \]
For $i = 0..5$
\[ P = Z[i] + P \]
If ($P \geq 0$)
\[ \text{Out}(P+T[0]+T[2]) \]
Else
\[ \text{Out}(-P+T[1]+T[3]) \]

\[ T[i] = (Y[i*2]+Y[i*2+1]) ^ 0.5 \]
Visualizing Buffer Analysis

Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

Index in Memory

Actor Firings

P = 0
for i = 0..5
    P = Z[i] + P
if (P >= 0)
    Out(P+T[0]+T[2])
else
    Out(-P+T[1]+T[3])

Z[i] = X[i] ^ 3

for i = 1..2
    X[i] = S[i] + 1
for i = 3..7
    Y[j] = S[i] ^ 2

T[i] = (Y[i*2] + Y[i*2 + 1]) ^ 0.5
Visualizing Buffer Analysis

Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

Index in Memory

Actor Firings

Y 7

0

Schedule:

Firing Sequence:
Visualizing Buffer Analysis

Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

\[ S \]

\[ A \]

\[ \text{for } i = 1..2 \]
\[ X[i] = S[i] + 1 \]
\[ \text{for } i = 3..7 \]
\[ Y[j] = S[i] \times 2 \]

\[ B \]

\[ Z[i] = X[i] \times 3 \]

\[ C \]

\[ P = 0 \]
\[ \text{for } i = 0..5 \]
\[ P = Z[i] + P \]
\[ \text{if } (P \geq 0) \]
\[ \text{Out}(P+T[0]+T[2]) \]
\[ \text{else} \]
\[ \text{Out}(-P+T[1]+T[3]) \]

\[ T[i] = (Y[i*2]+Y[i*2+1])^{0.5} \]

Index in Memory

Actor Firings

Y

0

7
Visualizing Buffer Analysis

Index in Memory

Actor Firings

Schedule:      2A    6B    4C D

Firing Sequence:        A   A   B   B   B   B   B   B   C   C   C C   D

\[ S \]

\[ A \]

\[ B = Z[i] = X[i] \land 3 \]

\[ C \]

\[ D \]

\[ P = 0 \]

\[ \text{for } i = 0..5 \]

\[ P = Z[i] + P \]

\[ \text{if } (P \geq 0) \]

\[ \text{Out}(P+T[0]+T[2]) \]

\[ \text{else} \]

\[ \text{Out}(-P+T[1]+T[3]) \]

\[ T[i] = (Y[i*2]+Y[i*2+1])^{0.5} \]
Visualizing Buffer Analysis

Schedule: 2A 6B 4C D
Firing Sequence: A A B B B B B C C C C D

\[
\begin{align*}
S & \quad A \\
& \quad B \\
& \quad C \\
& \quad D
\end{align*}
\]
\[
X[i] &= S[i] + 1 \\
Y[j] &= S[i] ^ 2 \\
Z[i] &= X[i] ^ 3 \\
T[i] &= (Y[i*2] + Y[i*2+1]) ^ 0.5
\]

Index in Memory

Actor Firings

\[
p = 0 \\
\text{for } i = 0..5 \\
\quad p = Z[i] + p \\
\text{if } (p \geq 0) \\
\quad \text{Out}(P+T[0]+T[2]) \\
\text{else} \\
\quad \text{Out}(-P+T[1]+T[3])
\]
Visualizing Buffer Analysis

Schedule: 2A 6B 4C D

Firing Sequence: A A B B B B B B C C C C D

```
for i = 1..2
X[i] = S[i] + 1
for i = 3..7
Y[j] = S[i] ^ 2

Z[i] = X[i] ^ 3
```

```
P = 0
for i = 0..5
  P = Z[i] + P
if (P >= 0)
  Out(P+T[0]+T[2])
else
  Out(-P+T[1]+T[3])
```

```
T[i] = (Y[i*2]+Y[i*2+1])^0.5
```

Index in Memory

Actor Firings

Y

7

0
Granularity and Buffer Allocation

- The granularity in buffer analysis compromises accuracy in temporal behavior of buffers with analysis complexity:
  - Baseline
  - Coarse-grain
  - Fine-grain
Granularity and Buffer Allocation

Baseline Analysis

Live Range Analysis
(Con-Grain)

Fine-Grain Analysis

Fine-Grain Buffer Allocation

- **Mathematic Formulation:**
  - Use of existing tools
  - Choose the best data structure

\[
\forall e \in E : B_e = (H_e, L_e)
\]

\(B_e\): The Buffer on edge \(e\) which we call it buffer \(e\) in short

\(H_e[t]\): Head index at time \(0 \leq t \leq T\) for the buffer on \(e\)

\(L_e[t]\): Tail index at time \(0 \leq t \leq T\) for the buffer on \(e\)

\[T = \sum_{v \in q_G} q[v]\]

\[O = \{(o_{e_1}, o_{e_2}, o_{e_3}, \ldots, o_{e_N}) \mid e_1 : e_N \in E, \ N = |E|}\]
**Fine-Grain Buffer Allocation**

- **LEMMA:**
  - In SA schedules the head index at the time $t$ is always greater than equal the tail index at the same time: $\forall t \leq T : H_e[t] \geq L_e[t]$

- **Constraints:**

  $$\forall e, b \in E \quad \forall 0 \leq t \leq T :$$
  $$H_e[t] + o_e \leq L_b[t] + o_b \quad OR \quad H_b[t] + o_b \leq L_e[t] + o_e$$

- **Objective:** Minimize **Shared Buffer Size**:

  $$SBS = \max_{\forall e \in E} \{ o_e + H_e^{max} \mid H_e^{max} = \max_{0 \leq t \leq T} (H_e[t]) \}$$
ILP Formulation

- The complexity of buffer sharing instance, and ILP runtime grows exponentially.
- Linear constraints cannot be easily used to articulate the “OR” logic:
  - Binary variables For each buffer and each location in the shared memory space
  - Constraints have to be generated for all time steps.
In several industries there is a need for packing a set of 2-dimensional objects on a larger rectangular unit of material by minimizing the waste.

- Two-Dimensional Bin Packing Problem (2BP):
  - wood or glass industries, warehousing contexts, newspapers paging

- Two-Dimensional Strip Packing Problem (2SP):
  - paper or cloth industries
The relationship between Packing Problems and Buffer Sharing Problem:

- Objects: Buffer Size in Time which form complex polygons
- Roll of Material: Shared Buffer Memory
- Objective: To allocate an index to each buffer in the shared memory with no conflict using minimum space
- Difference: We cannot move the objects (polygons) in time. We are only allowed to move them vertically. We also have no rotation.
MDA is moving down the buffers in the following order:

Move-Down Algorithm

- MDA is moving down the buffers in the following order:
Move-Down Algorithm

- MDA is moving down the buffers in the following order:
  
  \[
  \]
Move-Down Algorithm

- MDA is moving down the buffers in the following order:
The final placement of the buffers corresponding to the following order: G_H, D_G, C_D, B_C, A_B, E_G, C_E, F_G, C_F

The height of the final skyline indicates the shared memory size.
Another sequence which leads to the size 18 (14 is the optimal):

Evolutionary Optimization using MDA

- Genetic Algorithms in General:
  - **Chromosome**: Provides an abstract representation of solutions in the search space,
  - **Inheritance**: Models the basic operations through which, chromosomes are perturbed to improve the solution quality
    - Crossover
    - Mutation
  - **Fitness Function**: Quantizes the quality of candidate solutions, and determines survival of selected candidates.
Evolutionary Optimization using MDA

- Initialization: Randomly select a set of permutations

  \[ \text{Sample set} = \{\pi_1, \pi_2, \pi_3, \ldots, \pi_N\} \]

- Fitness function:

  \[ f(\pi) = \frac{1}{\text{height}(\pi)} \]

- Selection:

  \[ p(\pi_i) = \frac{f(\pi_i)}{\sum_{j=1}^{N} f(\pi_j)} \]
Evolutionary Optimization using MDA

- **Crossover:**
  - Example: \( p = 2 \quad q = 4 \)
    
    \[ \pi_{\text{parent}1} = (B_{e1}, B_{e2}, B_{e3}, B_{e4}, B_{e5}, B_{e6}) \]
    
    \[ \pi_{\text{parent}2} = (B_{e6}, B_{e5}, B_{e4}, B_{e3}, B_{e2}, B_{e1}) \]
    
    \[ \pi_{\text{child}} = (B_{e2}, B_{e3}, B_{e4}, B_{e6}, B_{e5}, B_{e1}) \]

- **Mutation:**
  - Example: \( p_{\text{mutation}} = 0.4 \) : the probability of being mutated
    
    \( i = 2 \quad j = 4 \)
    
    \[ \pi_{\text{child \ Before}} = (B_{e2}, B_{e3}, B_{e4}, B_{e6}, B_{e5}, B_{e1}) \]
    
    \[ \pi_{\text{child \ After}} = (B_{e2}, B_{e6}, B_{e4}, B_{e3}, B_{e5}, B_{e1}) \]

- Iteratively, new children are generated and compared to the existing members until the termination point where we can return the best solution found.
Experimental Evaluation

- We have integrated our algorithm into the MIT StreamIt compiler
- Three composite stream objects in StreamIt
- Filters specify data processing
Experimental Evaluation

- The StreamIt scheduler is designed based on the hierarchical nature of the language.
- In Split-joins, one large buffer is used to implement multiple channels that either split to or join from several actors.

S: 1( 5A 5B 4( 1C 1D 2( 1E 1F ) ) 10G 5H )
Experimental Evaluation

- **Benchmark Applications:**
  - Two sorting algorithms: Bitonic Sort, Insertion Sort
  - Two different implementation of the Fast Fourier Transform
  - Time Delay Estimation kernel
  - Matrix Multiplication kernel

<table>
<thead>
<tr>
<th></th>
<th>Number of Buffers</th>
<th>Number of Actores</th>
<th>Number of Time Steps</th>
<th>Baseline</th>
<th>Coarse-Grain</th>
<th>Fine-Grain (Best Case)</th>
<th>Fine-Grain (Worst Case)</th>
<th>Compile Time with GA in Sec.</th>
<th>Optimal Solution by ILP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitonic Sort</td>
<td>119</td>
<td>214</td>
<td>340</td>
<td>1152</td>
<td>96</td>
<td>48</td>
<td>64</td>
<td>91</td>
<td>32</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>8</td>
<td>9</td>
<td>263</td>
<td>1024</td>
<td>256</td>
<td>128</td>
<td>128</td>
<td>6</td>
<td>128</td>
</tr>
<tr>
<td>FFT2</td>
<td>22</td>
<td>24</td>
<td>446</td>
<td>3072</td>
<td>640</td>
<td>384</td>
<td>384</td>
<td>10</td>
<td>~</td>
</tr>
<tr>
<td>FFT3</td>
<td>38</td>
<td>64</td>
<td>175</td>
<td>960</td>
<td>192</td>
<td>72</td>
<td>96</td>
<td>11</td>
<td>64</td>
</tr>
<tr>
<td>TDE</td>
<td>48</td>
<td>51</td>
<td>17204</td>
<td>77120</td>
<td>23168</td>
<td>11776</td>
<td>23040</td>
<td>510</td>
<td>~</td>
</tr>
<tr>
<td>Matrix Mult.</td>
<td>10</td>
<td>21</td>
<td>2712</td>
<td>5000</td>
<td>4000</td>
<td>2000</td>
<td>2000</td>
<td>13</td>
<td>~</td>
</tr>
</tbody>
</table>
Experimental Evaluation

Improvement of coarse-grain and fine-grain methods compared to the baseline.
Experimental Evaluation

Improvement in all fine-grain cases: GA worst case, GA best case, and ILP, compared to the coarse-grain method
Conclusions

- Visualization of buffers transforms the allocation problem into packing of complex polygons.

- Fine-grain analysis vs. conventional coarse-grain live range analysis: dramatic improvements.

- The benefits of this approach outweighs the reasonable increase in static analysis latency for a large class of resource-constrained embedded systems.