Exact and Approximate Task Assignment Algorithms for Pipelined Software Synthesis

Matin Hashemi
Soheil Ghiasi

Laboratory for Embedded and Programmable Systems
http://leps.ece.ucdavis.edu

Department of Electrical and Computer Engineering
University of California, Davis
Davis, CA
United States
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]
}
```
Programming Model

- Thread-based models
  - Difficult to develop and debug [Sutter and Larus, ACM Queue ‘05]
  - Fundamentally, unreliable and nondeterministic [Lee, IEEE Computer ’06], [Weng, MIT tech report ’75]

- To maximize **throughput** of stream applications
  - Pipelined distributed-memory dual-core
  - Connected through on-chip network
Software Synthesis

- Need better CAD tools
  [Rowen, MPSOC ‘3], [Rabaey, Gigascale ’04], [Gordon, ASPLOS ’06], [Martin, DAC 06],
  [Parkhurst, ICCAD ’06], [Panel, EMSOFT ‘06], [Asanovic, UCB tech report ‘06]

- Need effective task assignment methods because of diminishing returns if applications don’t use available processing power
  [Leland, SC ’95], [Karypis, SC ’95], [Parkhurst, ICCAD ’06], [Martin, DAC 06],
  [Asanovic, UCB tech report ‘06]
Application Model: Dataflow 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.
- SDF is one special case
  - statically schedulable [Lee ‘87]
Example

N-Element Merge Sort
http://www.cag.csail.mit.edu/streamit
Example

Vocoder
http://www.cag.csail.mit.edu/streamit
Performance Model: Implementation Dependant

- Throughput can be any function of **workloads** and **communications**

- \( W_G \)
  - computation **workload**, unit time
  - estimated from source code
  - implementation dependant

- **Data rates**
  - # of data tokens
  - known at compile time [Lee ‘87]

- **C_{CUT}**
  - communication **cost**, unit time
  - implementation dependant
Performance Model: Example 1

```
while(1) {
    for i=1..n
        X[i] = S[i] + S[n-i]
        Y[i] = S[i] * S[i]
    for i=1..n
        Z[i] = q[i] * Y[n-i]
    for i=1..n
        writef(X[i])
        writef(Z[i])
}
```

```
while(1) {
    for i=1..n
        X[i] = readf()
        Y[i] = readf()
    for i=1..n
        Z[i] = q[i] * Y[n-i]
    for i=1..n
        T[i] = X[i] * Z[n-i]
}
```

1 / Throughput = Exec. Period = Max{ \( W_G^1 \), \( C_{CUT} \), \( C_{CUT} \), \( W_{G2} \) }

Correction factors for clock speeds
Performance Model:
Example 2

while(1) {
  for i=1..n
    X[i] = S[i]+S[n-i]
    writef(X[i])
  Y[i] = S[i]*S[i]
  for i=1..n
    Z[i] = q[i]*Y[n-i]
    writef(Z[i])
}

while(1) {
  for i=1..n
    X[i] = readf()
  for i=1..n
    Z[i] = readf()
  for i=1..n
    T[i] = X[i]*Z[n-i]
}

Exec. Period = Max{ \( W_1 + W_2 - 2N \times \text{mem} + N + N \), \( N + N + W_3 \) }
Performance Model:
Example 3

\[
\text{Exec. Period} = \text{Max}\{ W_1 + W_2 + OV + N + N, \ hop, N + N + OV + W_3 \}
\]

\[
\text{WG}_1 \quad \text{C}_{\text{CUT}} \quad \text{C}_{\text{CUT}} \quad \text{WG}_2
\]

\[
\begin{align*}
S & \quad X \quad Y \quad Z \\
\text{for } i = 1..N & \quad X_i = S_i + S_{N-i} \\
& \quad Y_i = S_i^2 \\
& \quad Z_i = q_i \cdot Y_{N-i}
\end{align*}
\]

\[
\begin{align*}
\text{for } i = 1..N & \quad T_i = X_i \cdot Z_{N-i} \\
\end{align*}
\]

\[
\begin{align*}
\text{while(1) \{} & \quad P[1..2n] = \text{readp}() \\
& \quad \text{for } i = 1..n \\
& \quad X[i] = P[i] \\
& \quad Z[i] = P[i+n] \\
& \quad \text{for } i = 1..n \\
& \quad P[i] = X[i] \\
& \quad P[i+n] = Z[i] \\
& \quad \text{writep}(P[1..2n]) \}\n\end{align*}
\]

\[
\begin{align*}
\text{while(1) \{} & \quad P[1..2n] = \text{readp}() \\
& \quad \text{for } i = 1..n \\
& \quad X[i] = P[i] \\
& \quad Z[i] = P[i+n] \\
& \quad \text{for } i = 1..n \\
& \quad T[i] = X[i] \cdot Z[n-i] \\
& \\}
\end{align*}
\]
Performance Model: Example 4

```
while(1) {
    for i=1..n
        X[i] = S[i] + S[n-i]
        writep(X[i])
    Y[i] = S[i] * S[i]
    for i=1..n
        Z[i] = q[i] * Y[n-i]
        writep(Z[i])
}
```

```
while(1) {
    for i=1..n
        X[i] = readp()
    for i=1..n
        Z[i] = readp()
    for i=1..n
        T[i] = X[i] * Z[n-i]
}
```

Exec. Period = Max{ \( W_1 + W_2 - 2N \times \text{mem} + N + N \), (N+N)hop, N+N+W_3 }
Versatile Cost Function

- Throughput = 1 / Execution Period
  - implementation dependant
- Task assignment method has to be versatile: handle any realistic hardware-inspired function of
  - workloads
  - communications
- $Q_{\text{CUT}} = F (W_{G1}, C_{\text{CUT}}, W_{G2})$
- realistic: $Q_{\text{CUT}}$ has to be non-decreasing in $C_{\text{CUT}}$
Convex Cut

- To ensure a feasible schedule
  [Cong, FPGA ’07]
  - we need all data at the beginning

```plaintext
while(1) {
    for i=1..n
        A[i] = readf()
    for i=1..n
        Y[i] = readf()
    // computation
}
```

- Cycles limit the throughput
  [Rabaey ’93], [Wolf ’94]
Algorithm Idea

- Calculate cost function $Q_C$ only from cut $C$, and not other parts of the graph
- Move workloads to edges
- Property:
  - $W_C = (0) + (W_e) + (W_c) + (W_a + W_b) = W_a + W_b + W_c + W_e = W_{G1}$
  - $C_C = C_{cf} + C_{eg} + C_{cd} + C_{bd}$
Algorithm Details

- move node workloads of G to its edges
- for planar graphs, a cut is equal to a path in dual graph
- expand $G^*$ to $G'$

$W_G = 3 + 1 + 1 + 2$
Algorithm Details, cont.

- single-source shortest-path on G’
- pick the best cut

\[ s^* \rightarrow t^* \quad m^* \quad W_c = W_{G1} \]

<table>
<thead>
<tr>
<th>( W_{G1} )</th>
<th>( C_C )</th>
<th>shortest path for this ( W_{G1} )?</th>
<th>( W_{G2} ) = ( W_G - W_{G1} )</th>
<th>cost function ( Q_C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1+4 =5</td>
<td>✓</td>
<td>7-3=4</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>1+2 =3</td>
<td>✓</td>
<td>7-4=3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>3+4 =7</td>
<td></td>
<td>7-4=3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3+2 =5</td>
<td>✓</td>
<td>7-5=2</td>
<td>10</td>
</tr>
</tbody>
</table>

provably optimal in minimizing any realistic cost function

Max \{\( W_{G1} + C_C \), \( W_{G2} + C_C \}\}
Complexity

- Constructing $G'$ is the most complex part of the algorithm.
- Both runtime and memory consumption depend on the number of vertices in $G'$.
- $O(N \times W_G)$
- NP Complete: reduction from set partitioning
Approximate Algorithm

- Approximate workload values in graph G’. A range of workload values $w$ is represented by one single $y$ value, where $y=f(w)$ is the approximation function, and $\delta$ is a constant parameter:

\[ f(w) = (1 + \delta) \left\lfloor \log_{1+\delta} w \right\rfloor \]

- Example (1+\delta=2)

<table>
<thead>
<tr>
<th>w</th>
<th>y=f(w)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2-3</td>
<td>2</td>
</tr>
<tr>
<td>4-7</td>
<td>4</td>
</tr>
<tr>
<td>8-15</td>
<td>8</td>
</tr>
</tbody>
</table>
Approximate Algorithm, cont.

- Theorem:
  \[
  \frac{w}{1+\delta} < y \leq w
  \]
  \[
  \frac{W_C}{(1+\delta)^k} < \Psi_C \leq W_C
  \]

- Cost function
  - \( Q_C = F (W_{G1}, C_C, W_{G2}) \)
  - \( \Omega_C = F (\Psi_{G1}, C_C, \Psi_{G2}) \)
  - \( Q_{C,\text{min}} < \Omega_{C,\text{min}} < (1+\varepsilon) Q_{C,\text{min}} \)
  - \( \varepsilon = \delta k \)

- Error in calculating cost function is bounded within an adjustable factor.

<table>
<thead>
<tr>
<th>w</th>
<th>( y = f(w) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2-3</td>
<td>2</td>
</tr>
<tr>
<td>4-7</td>
<td>4</td>
</tr>
<tr>
<td>8-15</td>
<td>8</td>
</tr>
</tbody>
</table>
Experiment Platform

- Digilent Virtex II PRO board
- Processors: MicroBlaze
- Communication links: FSL

![](image)

- Application ( .str file )

StreamIt Compiler

- Task Scheduling
- Task Assignment
- Processor Assignment
- Code Generation

Xilinx EDK

C Compiler

Measurement on FPGA Board

binary code

partitioned application ( multiple .c files )

application ( .str file )
**StreamIt**

- basic element: **Filter**
- constructs:
  - Pipeline, SplitJoin, Feedback
- planar graph

**Partitioning Algorithm:**

[Thies, MIT tech report ‘03]
- limited to structured graphs
- dynamic programming

\[ B = 9 + 3 + 2 + 3 \]
\[ \sum_{b \in B} X_b Y_b = O( N^2 B ) \]
## Benchmarks

<table>
<thead>
<tr>
<th>Description</th>
<th>Task Graph Structure</th>
<th>Single-processor Throughput (K sample / sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BSORT</strong></td>
<td>Bitonic Sort</td>
<td>756</td>
</tr>
<tr>
<td><strong>MATMUL</strong></td>
<td>Blocked Matrix Multiply</td>
<td>23</td>
</tr>
<tr>
<td><strong>FFT</strong></td>
<td>Fast Fourier Transform</td>
<td>152</td>
</tr>
<tr>
<td><strong>TDE</strong></td>
<td>Frequency Domain Convolution</td>
<td>46</td>
</tr>
<tr>
<td><strong>FILTER</strong></td>
<td>Discrete Filter</td>
<td>53</td>
</tr>
</tbody>
</table>
## Throughput

<table>
<thead>
<tr>
<th>Workload Imbalance %</th>
<th>Throughput</th>
<th>Workload Imbalance %</th>
<th>Throughput</th>
<th>Throughput vs single-processor</th>
<th>Additional Throughput (TAP-StreamIt)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BSORT</strong></td>
<td>7.7</td>
<td>296.3</td>
<td>3.7</td>
<td>319.2</td>
<td>1.58</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>.12</strong></td>
</tr>
<tr>
<td><strong>MATMUL</strong></td>
<td>6.5</td>
<td>186.3</td>
<td>9.7</td>
<td>208.0</td>
<td>1.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.55</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>.17</strong></td>
</tr>
<tr>
<td><strong>FFT</strong></td>
<td>11</td>
<td>417.7</td>
<td>4.9</td>
<td>470.4</td>
<td>1.58</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.77</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>.19</strong></td>
</tr>
<tr>
<td><strong>TDE</strong></td>
<td>4.1</td>
<td>933.8</td>
<td>4.1</td>
<td>933.8</td>
<td>1.61</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.61</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>.00</strong></td>
</tr>
<tr>
<td><strong>FILTER</strong></td>
<td>0.7</td>
<td>34.6</td>
<td>0.7</td>
<td>34.6</td>
<td>1.88</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.88</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>.00</strong></td>
</tr>
<tr>
<td><strong>Avg.</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.61</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>.09</strong></td>
</tr>
</tbody>
</table>

Throughput of dual-processor hardware for both StreamIt and TAP algorithms.
FFT
StreamIt / TAP

39%
45%
55%
61%
MATMUL
StreamIt / TAP

57%  57%
60%  43%
40%
TDE
StreamIt / TAP
## Exact Algorithm

<table>
<thead>
<tr>
<th></th>
<th>Runtime (second)</th>
<th>Memory Consumption (MB)</th>
<th>Dual-processor Throughput (K sample/sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BSORT</strong></td>
<td>31.8</td>
<td>2543</td>
<td>319</td>
</tr>
<tr>
<td><strong>MATMUL</strong></td>
<td>57.5</td>
<td>321</td>
<td>208</td>
</tr>
<tr>
<td><strong>FFT (64)</strong></td>
<td>46.7</td>
<td>2553</td>
<td>470</td>
</tr>
<tr>
<td><strong>TDE</strong></td>
<td>76.6</td>
<td>844</td>
<td>933</td>
</tr>
<tr>
<td><strong>FILTER</strong></td>
<td>121.5</td>
<td>1366</td>
<td>34.6</td>
</tr>
</tbody>
</table>

Throughput of dual-processor hardware when using the exact partitioning algorithm. It requires the mentioned time and memory to perform.
Approximate Algorithm

Throughput degradation versus reduction in runtime and memory consumptions when using the approximate partitioning algorithm. All values are normalized against the exact algorithm.
questions?