Optimal 2D Data Partitioning for DMA Transfers on MPSoCS

S. Saidi\textsuperscript{1,2} P. Tendulkar \textsuperscript{1} T. Lepley\textsuperscript{2} O. Maler\textsuperscript{1}

\textsuperscript{1}Verimag

\textsuperscript{2}STMicroelectronics

DSD 2012
1. Context and Motivation

2. Independent Computations

3. Overlapped Data Computations

4. Experiments
Outline

1. Context and Motivation
2. Independant Computations
3. Overlapped Data Computations
4. Experiments
Motivation

- How to reduce/hide the off-chip memory latency?
Heterogeneous Multi-core Architectures

- a powerful host processor and a multi-core fabric to accelerate computationally heavy kernels.
Heterogeneous Multi-core Architectures

- a powerful host processor and a multi-core fabric to accelerate computationally heavy kernels.
Offloadable kernels work on large data sets, initially stored in the off-chip memory.

Algorithm

\[
\text{for } i_1 = 1 \text{ to } n_1 \\
\text{ for } i_2 = 1 \text{ to } n_2 \\
\quad Y[i_1, i_2] = f(X[i_1, i_2]) \\
\text{ od}
\]
Data Transfers

- High off-chip memory latency: accessing off-chip data is very costly

```
Algorithm
for i1 = 1 to n1
  for i2 = 1 to n2
    Y[i1, i2] = f(X[i1, i2])
  od
```

![Diagram of data transfers and memory architecture](image)

Saidi, Tendulkar, Lepley, Maler
Optimizing DMA Transfers
**Data Transfers**

- **High off-chip memory latency:** accessing off-chip data is very costly

```plaintext
Algorithm
for i_1 = 1 to n_1
  for i_2 = 1 to n_2
    Y[i_1, i_2] = f(X[i_1, i_2])
  od
```

Saidi, Tendulkar, Lepley, Maler
Optimizing DMA Transfers
8 / 49
Data Partitioning:

- $s_1$: block height
- $s_2$: block width

Granularity of transfers: rectangular clusters of arrays elements.

Sequential execution of computations and data transfers.

Saidi, Tendulkar, Lepley, Maler

Optimizing DMA Transfers
DMA Data Transfers

Granularity of transfers: rectangular clusters of \((s_1, s_2)\) arrays elements,

Data Partitioning:

\[
\begin{align*}
X(i_1, i_2) = & \quad \text{block}\left(j_1, j_2\right) \\
\text{Fetch}(\text{block}_i) \quad & \quad \text{Compute}(\text{block}_i) \quad \text{Write back}(\text{block}_i) \\
\text{while } (i < (m_1 \times m_2)) & \quad i++
\end{align*}
\]
DMA Data Transfers

Granularity of transfers: rectangular clusters of \((s_1, s_2)\) arrays elements,

Data Partitioning:

- **DMA get**
  - Fetch\((\text{block}_i)\)
  - Compute\((\text{block}_i)\)
  - Write back\((\text{block}_i)\)

while \((i < (m_1 \times m_2))\)

\[ i++ \]

\[ i = 1 \]
DMA Data Transfers

Granularity of transfers: rectangular clusters of \((s_1, s_2)\) arrays elements,

Data Partitioning:

\[
X(i_1, i_2) = \text{block}(j_1, j_2)
\]

\[
\text{while } (i < (m_1 \times m_2)) \quad i++
\]

\[
\text{Compute(block}_i) \quad \text{Fetch(block}_i) \quad \text{Write back(block}_i)
\]
DMA Data Transfers

Granularity of transfers: rectangular clusters of \((s_1, s_2)\) arrays elements,

Data Partitioning:

\[
X(i_1, i_2) = \text{block}(j_1, j_2)
\]

Sequential execution of computations and data transfers.

\[
i = 1
\]

while \(i < (m_1 \times m_2)\)

\[
i++
\]

\[
\text{Fetch(block}_i) \quad \text{Compute(block}_i) \quad \text{Write back(block}_i) \quad \text{DMA}_\text{put}
\]
DMA Data Transfers

Granularity of transfers: rectangular clusters of \((s_1, s_2)\) arrays elements,

Data Partitioning:

\[
\text{block}(j_1, j_2)
\]

\[
X(i_1, i_2)
\]

\[
\begin{align*}
\text{DMA}_\text{put} & \quad i = 1 \\
\text{DMA}_\text{put} & \quad \text{Fetch(block}_i) \\
\text{DMA}_\text{put} & \quad \text{Compute(block}_i) \\
\text{DMA}_\text{put} & \quad \text{Write back(block}_i) \\
\end{align*}
\]

while \((i < (m_1 \times m_2))\)

\[
\begin{align*}
\text{DMA}_\text{put} & \quad i++
\end{align*}
\]

- Sequential execution of computations and data transfers.
Context and Motivation

Software Pipelining

Asynchronous DMA calls and double buffering:

\[
\begin{align*}
&\text{Fetch(}block_0) \\
&\text{dma.get(}local - buffer[1], block_0, s) \\
&i = 0 \\
&\text{while } (i < (n/s) - 1) \\
&i++ \\
&\text{Compute(}block_i) \\
&\text{dma.get(}local - buffer[2], block_{i+1}, s) \\
&\text{Write back(}block_i) \\
&\text{Compute(}block_{(n/s) - 1}) \\
&\text{Write back(}block_{(n/s) - 1}) \\
&\text{Fetch(}block_{i+1})
\end{align*}
\]
Software Pipelining

Asynchronous DMA calls and double buffering:

while \((i < (n/s) - 1)\)

\[ i = 0 \]

\[ i + + \]

\( DMA \_get(local \_buffer[1], block_0, s) \)

\( DMA \_get(local \_buffer[2], block_{i+1}, s) \)

\( Fetch(block_0) \)

\( Compute(block_i) \)

\( Write \_back(block_i) \)

\( Compute(block_{(n/s) - 1}) \)

\( Write \_back(block_{(n/s)-1}) \)

\( Fetch(block_{i+1}) \)
Context and Motivation

Software Pipelining

Asynchronous DMA calls and double buffering:

\[ \text{Compute(block}_i\text{)} \quad \text{Fetch(block}_{i+1}\text{)} \]

\[ \text{While } (i < (n/s) - 1) \]

\[ i++ \]

\[ \text{Compute(block}_{(n/s) - 1}\text{)} \]

\[ \text{Write back(block}_i\text{)} \]

\[ \text{Write back(block}_{(n/s) - 1}\text{)} \]

\[ \text{DMA get(local - buffer}[1], block_0, s) \]

\[ \text{DMA get(local - buffer}[2], block}_{i+1}, s) \]
Software Pipelining

Asynchronous DMA calls and double buffering:

\[\begin{align*}
\text{while } (i < \left(\frac{n}{s}\right) - 1) & \\
& \quad i++
\end{align*}\]

\[\begin{align*}
\text{Compute}(& \text{block}_i) \\
\text{Write back}(& \text{block}_i) \\
\text{Compute}(& \text{block}_{\left(\frac{n}{s}\right) - 1}) \\
\text{Write back}(& \text{block}_{\left(\frac{n}{s}\right) - 1})
\end{align*}\]

\[\begin{align*}
\text{Fetch}(& \text{block}_0) \\
\text{dma.get}(& \text{local - buffer}[1], \text{block}_0, s) \\
\text{Fetch}(& \text{block}_{i+1}) \\
\text{dma.get}(& \text{local - buffer}[2], \text{block}_{i+1}, s)
\end{align*}\]
Software Pipelining

**Asynchronous DMA calls and double buffering:**

\[
\begin{align*}
&\text{Fetch(block}_0) \\
&\text{Compute(block}_i) \\
&\text{Write back(block}_i) \\
&\text{Compute(block}_{(n/s) - 1}) \\
&\text{Write back(block}_{(n/s) - 1}) \\
&\text{dma}_{\text{get}}(\text{local} - \text{buffer}[1], \text{block}_0, s) \\
&\text{dma}_{\text{get}}(\text{local} - \text{buffer}[2], \text{block}_{i+1}, s) \\
&i = 0 \\
&\text{while } (i < (n/s) - 1) \text{ do} \\
&\quad i++
\end{align*}
\]
Software Pipelining

Overlap of,

- **Computation of current block,**
- **Transfer of next block.**

Optimal Granularity:
What is the choice of \((s^*, s^*_{\text{opt}})\) that optimizes performance?
Software Pipelining

Overlap of,

- *Computation of current block*,
- *Transfer of next block*.

![Diagram of computation and transfer overlap]

Optimal Granularity:

What is the choice of \((s_1^*, s_2^*)\) that optimizes performance?
Software Pipelining

Overlap of,

- *Computation* of current block,
- *Transfer* of next block.
Software Pipelining

Overlap of,

- *Computation* of current block,
- *Transfer* of next block.

Optimal Granularity:

What is the choice of \((s_1^*, s_2^*)\) that optimizes performance?
Software Pipelining

Overlap of,

- *Computation* of current block,
- *Transfer* of next block.
Software Pipelining

Overlap of,

- *Computation of* current block,
- *Transfer of* next block.

```
R1 -> R2 -> R3 -> R4 -> ... -> Rm
|
C1 -> C2 -> C3 -> ... -> Cm-1 -> Cm
|
W1 -> W2 -> ... -> Wm-2 -> Wm-1 -> Wm
```

Prologue     ____     ____     ____     ____     ____     ____

Epilogue
**Software Pipelining**

- We want to optimize execution of the pipeline.

![Diagram of the pipeline](image)
Software Pipelining

- We want to optimize execution of the pipeline.

Optimal Granularity:
What is the choice of \((s_1^*, s_2^*)\) that optimizes performance?
Computation and Transfer Regimes

- $T(s_1, s_2)$ and $C(s_1, s_2)$: Transfer and Computation time of a block

Transfer Regime $T > C$:

Computation Regime $C > T$:
Computation and Transfer Regimes

- $T(s_1, s_2)$ and $C(s_1, s_2)$: Transfer and Computation time of a block

**Transfer Regime $T > C$:**

- **Input transfer**
  - $b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8$

- **Computation**
  - $b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8$

- **Output transfer**
  - $b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8$

**Computation Regime $C > T$:**

- **Input transfer**
  - $b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8$

- **Computation**
  - $b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8$

- **Output transfer**
  - $b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8$
**Computation and Transfer Regimes**

- \( T(s_1, s_2) \) and \( C(s_1, s_2) \): Transfer and Computation time of a block

### Transfer Regime \( T > C \):

<table>
<thead>
<tr>
<th>Input transfer</th>
<th>Computation</th>
<th>Output transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_0 )</td>
<td>( b_0 )</td>
<td>( b_0 )</td>
</tr>
<tr>
<td>( b_1 )</td>
<td>( b_1 )</td>
<td>( b_1 )</td>
</tr>
<tr>
<td>( b_2 )</td>
<td>( b_2 )</td>
<td>( b_2 )</td>
</tr>
<tr>
<td>( b_3 )</td>
<td>( b_3 )</td>
<td>( b_3 )</td>
</tr>
<tr>
<td>( b_4 )</td>
<td>( b_4 )</td>
<td>( b_4 )</td>
</tr>
<tr>
<td>( b_5 )</td>
<td>( b_5 )</td>
<td>( b_5 )</td>
</tr>
<tr>
<td>( b_6 )</td>
<td>( b_6 )</td>
<td>( b_6 )</td>
</tr>
<tr>
<td>( b_7 )</td>
<td>( b_7 )</td>
<td>( b_7 )</td>
</tr>
<tr>
<td>( b_8 )</td>
<td>( b_8 )</td>
<td>( b_8 )</td>
</tr>
</tbody>
</table>

### Computation Regime \( C > T \):

<table>
<thead>
<tr>
<th>Input transfer</th>
<th>Computation</th>
<th>Output transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_0 )</td>
<td>( b_0 )</td>
<td>( b_0 )</td>
</tr>
<tr>
<td>( b_1 )</td>
<td>( b_1 )</td>
<td>( b_1 )</td>
</tr>
<tr>
<td>( b_2 )</td>
<td>( b_2 )</td>
<td>( b_2 )</td>
</tr>
<tr>
<td>( b_3 )</td>
<td>( b_3 )</td>
<td>( b_3 )</td>
</tr>
<tr>
<td>( b_4 )</td>
<td>( b_4 )</td>
<td>( b_4 )</td>
</tr>
<tr>
<td>( b_5 )</td>
<td>( b_5 )</td>
<td>( b_5 )</td>
</tr>
<tr>
<td>( b_6 )</td>
<td>( b_6 )</td>
<td>( b_6 )</td>
</tr>
<tr>
<td>( b_7 )</td>
<td>( b_7 )</td>
<td>( b_7 )</td>
</tr>
<tr>
<td>( b_8 )</td>
<td>( b_8 )</td>
<td>( b_8 )</td>
</tr>
</tbody>
</table>
Find \((s_1^*, s_2^*)\) such that,

\[
\begin{align*}
\min & \quad T(s_1, s_2) \\
\text{s.t.} & \quad T(s_1, s_2) \leq C(s_1, s_2) \\
( & \quad (s_1, s_2) \in [1..n_1] \times [1..n_2] \\
& \quad s_1 \times s_2 \leq M
\end{align*}
\]
Our Contribution

We derive optimal granularity for 2D\(^1\) DMA transfers,

1. Independant Data Computations.
2. Overlapped Data Computations.

\(^1\)1D data work was published in Hipecac 2012,

Outline

1. Context and Motivation

2. Independent Computations

3. Overlapped Data Computations

4. Experiments
Characterization of Computation and Transfer Time:

\((s_1 \times s_2)\): nb array elements clustered in one block,

\[ \text{Computation time } C(s_1, s_2) : \]

\[ \omega : \text{time to compute one element,} \]

\[ C(s_1, s_2) = \omega \cdot s_1 \cdot s_2 \]

Diagram:
- Block \((j_1, j_2)\)
- \(s_1\) and \(s_2\) dimensions
- \(X(i_1, i_2)\) element index
Characterization of Computation and Transfer Time:

\((s_1 \times s_2)\): nb array elements clustered in one block,

Computation time \(C(s_1, s_2)\):

\[ C(s_1, s_2) = \omega \cdot s_1 \cdot s_2 \]

\(\omega\): time to compute one element,
Characterization of Computation and Transfer Time:

\((s_1 \times s_2)\): nb array elements clustered in one block,

\[
C(s_1, s_2) = \omega \cdot s_1 \cdot s_2
\]

Computation time \(C(s_1, s_2)\):

- \(\omega\): time to compute one element,

\[
C(s_1, s_2) = \omega \cdot s_1 \cdot s_2
\]
Contiguous DMA Transfers

DMA Transfer time $T(s_1, s_2)$:
Contiguous DMA Transfers

DMA Transfer time $T(s_1, s_2)$:

- $l$: fixed DMA initialization cost,
Contiguous DMA Transfers

DMA Transfer time $T(s_1, s_2)$:

- $I$: fixed DMA initialization cost,
- $\alpha$: transfer cost per byte, ($\alpha_p$ with multiple processors.)
Contiguous DMA Transfers

DMA Transfer time $T(s_1, s_2)$:

- $l$: fixed DMA initialization cost,
- $\alpha$: transfer cost per byte, ($\alpha_p$ with multiple processors.)
- $b$: size of one array element,
Contiguous DMA Transfers

DMA Transfer time $T(s_1, s_2)$:

- $l$: fixed DMA initialization cost,
- $\alpha$: transfer cost per byte, ($\alpha_p$ with multiple processors.)
- $b$: size of one array element,

$$T(s_1, s_2) = l + \alpha \cdot b(s_1 \cdot s_2)$$
### Strided DMA Transfers

DMA Transfer time $T(s_1, s_2)$:

- $s_1$: vertical stride
- $s_2$: horizontal stride
Strided DMA Transfers

DMA Transfer time $T(s_1, s_2)$:

$$T(s_1, s_2) = \tau + \alpha \cdot b(s_1 \cdot s_2)$$
Strided DMA Transfers

DMA Transfer time $T(s_1, s_2)$:

- $l_1$: transfer cost overhead per line,

$$T(s_1, s_2) = l + l_1 s_1 + \alpha \cdot b(s_1 \cdot s_2)$$

Strided DMA transfers are costlier than contiguous transfers
Computation Transfer Ratio

- $C(s_1, s_2)$: computation time of a block,
- $T(s_1, s_2)$: transfer time of a block,
Computation Transfer Ratio

- $C(s_1, s_2)$: computation time of a block,
- $T(s_1, s_2)$: transfer time of a block,
Optimal Granularity

**Pb Formulation**

\[ \text{Min } T(s_1, s_2) \text{ s.t. } \]

\[ T(s_1, s_2) \leq C(s_1, s_2) \]

\[ (s_1, s_2) \in [1..n_1] \times [1..n_2] \]

\[ s_1 \times s_2 \leq M \]

- \( s_1 \): block height
- \( s_2 \): block width
Optimal Granularity

Pb Formulation

Min $T(s_1, s_2)$ s.t.

$T(s_1, s_2) \leq C(s_1, s_2)$

$(s_1, s_2) \in [1..n_1] \times [1..n_2]$

$s_1 \times s_2 \leq M$

- $s_1$: block height
- $s_2$: block width
Optimal Granularity

\[
\begin{align*}
    T &= C \\
    n_1 &= 1 \\
    n_2 &= (1/\psi)(l_1 + l_0)
\end{align*}
\]

Optimal granularity is the Contiguous block to reach the computation regime:
Local Memory Constraint
Local Memory Constraint

\[ \frac{M}{s_2} T = C \]

\[ n_1 \]

\[ 1 \]

\[ s_1 \]

\[ s_2 \]

\[ M/b \]

\[ n_2 \]
Outline

1. Context and Motivation
2. Independant Computations
3. Overlapped Data Computations
4. Experiments
Shared Data

- Data parallel loop with shared input data:

\[
\begin{align*}
\text{for } i &:= 1 \text{ to } n_1 \text{ do} \\
&\text{for } i := 2 \text{ to } n_2 \text{ do} \\
Y[i_1, i_2] &:= f(X[i_1, i_2], V[i_1, i_2]); \\
V[i_1, i_2] &= \{X[i_1 - 1, i_2], X[i_1, i_2 - 1], \\
&\ldots, X[i_1 - k, i_2]\}
\end{align*}
\]

- symmetric window,

We consider replication at each block transfer.
Replicated Area and Transfer Cost

- Consider $s_1 \times s_2$ fixed,

\[
(s_1, s_2) = (2, 2) \quad (s_1, s_2) = (1, 4) \quad (s_1, s_2) = (4, 1)
\]
Overlapped Data Computations

Replicated Area and Transfer Cost

- Consider $s_1 \times s_2$ fixed,
- Compare Transfer cost of a flat and a square block,

\[
R = 12
\]

\[
( s_1, s_2 ) = ( 2, 2 )
\]

\[
R = 14
\]

\[
( s_1, s_2 ) = ( 1, 4 )
\]

\[
R = 14
\]

\[
( s_1, s_2 ) = ( 4, 1 )
\]
Replicated Area and Transfer Cost

- Consider $s_1 \times s_2$ fixed,
- Compare Transfer cost of a flat and a square block,

\[
R = 12
\]

\[
R = 14
\]

- More transfer lines Overhead

\[ (s_1, s_2) = (2, 2) \]

\[ (s_1, s_2) = (1, 4) \]

\[ (s_1, s_2) = (4, 1) \]
Replicated Area and Transfer Cost

- Consider $s_1 \times s_2$ fixed,
- Compare Transfer cost of a flat and a square block,

\[
\begin{align*}
R &= 12 \\
(s_1, s_2) &= (2, 2)
\end{align*}
\]

\[
\begin{align*}
R &= 14 \\
(s_1, s_2) &= (1, 4)
\end{align*}
\]

\[
\begin{align*}
R &= 14 \\
(s_1, s_2) &= (4, 1)
\end{align*}
\]

- More transfer lines Overhead
- More Replicated data Overhead
Replicated Area and Transfer Cost

- Consider $s_1 \times s_2$ fixed,
- Compare Transfer cost of a flat and a square block,

\[
R = 12 \\
(s_1, s_2) = (2, 2)
\]

\[
R = 14 \\
(s_1, s_2) = (1, 4)
\]

- More transfer lines Overhead
- More Replicated data Overhead
- More Replicated data and transfer lines Overheads
Replicated Area and Transfer Cost

More transfer lines overhead
More replicated data overhead
More replicated data and transfer lines overhead

Granularity Choice:
There is a tradeoff to find between the 2 overheads.
Optimal Granularity: Problem Formulation

Find \((s_1^*, s_2^*)\) such that,

\[
\min \ T(s_1 + k, s_2 + k) \quad \text{s.t.}
\]

\[
T(s_1 + k, s_2 + k) \leq C(s_1, s_2)
\]

\[(s_1, s_2) \in [1..n_1] \times [1..n_2]
\]

\[(s_1 + k) \times (s_2 + k) \leq M
\]
Double Buffering Optimal Granularity

\[ T = C \quad T_k = C \]

\[ s_1 \]

\[ n_1 \]

\[ s_2 \]

\[ n_2 \]

\[ s_1 = s_2 \]

\[ s^* \]
Double Buffering Optimal Granularity

\[
\begin{align*}
    T &= C \\
    T_k &= C
\end{align*}
\]

\[
\left\{ \begin{array}{l}
    s_1^* = \Delta + \left( \frac{c_1}{\psi} \right)(1/D) \\
    s_2^* = \Delta + \left( \frac{l_1}{\psi} \right)(1 + D)
\end{array} \right.
\]
Outline

1. Context and Motivation
2. Independant Computations
3. Overlapped Data Computations
4. Experiments
Platform Characteristics:

- 9-core heterogeneous multi-core architecture, with a Power Processor Element (PPE) and 8 Synergistic Processing Elements (SPE).
- Limited local store capacity per SPE: 256 Kbytes
- Explicitly managed memory system, using DMAs
Measured DMA Latency

- **Profiled hardware parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Cost (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA issue time</td>
<td>$l_0$</td>
<td>$\approx 108$</td>
</tr>
<tr>
<td>DMA overhead per line</td>
<td>$l_1$</td>
<td>$\approx 50$</td>
</tr>
<tr>
<td>Off-chip memory transfer cost/byte: 1 proc</td>
<td>$\alpha(1)$</td>
<td>$\approx 2.57$</td>
</tr>
<tr>
<td>Off-chip memory transfer cost/byte: $p$ proc</td>
<td>$\alpha(p)$</td>
<td>$\approx p \cdot \alpha(1)$</td>
</tr>
</tbody>
</table>
Predicted and Measured Optimal Granularity

- We implement double buffering on a mean filtering algorithm,
- $\omega = 62$ clock cycles.
- size of shared data: $(s_1 + 8) \times (s_2 + 8)$
We presented a general methodology for automating decisions about, **optimal block size and shape** for data transfers.

We validated the experiments on the Cell architecture.

**On-going Work and Perspectives:**

1. Extend the work to other platforms, like P2012,
2. Consider computation variabilities.
3. Combine task and data parallelism.