Optimizing DMA Data Transfers for Embedded Multi-Cores

Selma Saïdi

Jury members:

Oded Maler: Dir. de these
Luca Benini: Rapporteur
Eric Flamand: Examineur

Ahmed Bouajjani: President du Jury
Albert Cohen: Rapporteur
Bruno Jego: Examineur
Context of the Thesis

- Ph.D CIFRE with STMicroelectronics, supervised by,
  - Oded Maler, Verimag,
  - Bruno Jego and in collaboration with Thierry Lepley, STMicroelectronics
- Minalogic project ATHOLE
  - low-power multi-core platform for embedded systems
  - partners: ST, CEA, Thales, CWS, Verimag
Outline

1. Context and Motivation

2. Contribution
   - Problem Definition
   - Optimal Granularity for a Single Processor
     - 1Dim Data
     - 2Dim Data
   - Multiple Processors
   - Shared Data

3. Experiments on the Cell.BE

4. The move towards Platform 2012

5. Conclusions and Perpectives
Embedded Systems

There is an increasing requirement for performance under low power constraints:

- Need to integrate more functionalities in Embedded devices,
- Applications are becoming more computationally intensive and power hungry,
Running 2 processors in the same chip at half the speed will be less energy consuming and equally performant,
Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
- main characteristic: explicitly managed Memories:

features:
1. scratchpad memories and not caches,
2. data movement are explicitly managed by the software,
3. typically using a DMA (Direct Memory Access) engine: a hardware accelerator for managing transfers.
Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
- Main characteristic: explicitly managed Memories:

Features:

1. Scratchpad memories (No caches),
Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
- main characteristic: explicitly managed Memories:

Features:
1. Scratchpad memories (No caches)
2. DMA engine: a hardware for managing data transfers,
Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
- main characteristic: explicitly managed Memories:

Features:

1. Scratchpad memories (No caches)
2. DMA engine: a hardware for managing data transfers,
Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
- main characteristic: explicitly managed Memories:

Features:
1. **Scratchpad memories** (No caches)
2. **DMA engine**: a hardware for managing data transfers,
3. data transfers are explicitly managed by the software,
Embedded Multicore Architectures:

- Platform 2012: a manycore computation fabric,
- main characteristic: explicitly managed Memories:

Features:
1. Scratchpad memories (No caches)
2. DMA engine: a hardware for managing data transfers,
3. data transfers are explicitly managed by the software,
Embedded Multicore Architectures:

- acts as a general purpose programmable accelerator:
  ⇒ Heterogeneous Multicore Architectures,
Embedded Multicore Architectures:

- acts as a general purpose programmable accelerator:
  ⇒ Heterogeneous Multicore Architectures,
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 a distant off-chip memory.

\[
\text{Algorithm} \quad \text{for} \quad i = 0 \ \text{to} \ n - 1 \\
Y[i] = f(X[i]) \\
\text{od}
\]
Heterogeneous Multi-core Architectures

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

Algorithm

\[
\text{for } i = 0 \text{ to } n - 1
\]
\[
Y[i] = f(X[i])
\]

\[X \rightarrow T_0 \rightarrow P_{E_0} \rightarrow Y\]
Heterogeneous Multi-core Architectures

Data is transferred to a closer but smaller on-chip memory, using DMAs (Direct Memory Access).

```
Algorithm
for i = 0 to n - 1
    Y[i] = f(X[i])
    od
```

```
T0

PE0

block 0

... block 1

Multi-core fabric

Host CPU

Data Block Transfers

Off-chip Memory

Memory

DMA

Interconnect

Memory

PEn

X

Y
```
DMA Data Transfers: Single Buffering

$s$: number of array elements in one block,

\[
\begin{array}{cccccccccc}
\text{block}_0 & & \text{block}_1 & & \text{block}_{m-2} & & \text{block}_{m-1} \\
\hline
x[0] & x[1] & & & & & & & x[n-1] \\
\hline
s & & n & & & & & & \\
\end{array}
\]
DMA Data Transfers: Single Buffering

$s$: number of array elements in one block,

Sequential execution of computations and data transfers.

while ($i < n/s$)

$i++$

Fetch($block_i$)

Compute($block_i$)

Write back($block_i$)
DMA Data Transfers: Single Buffering

$s$: number of array elements in one block,

\[ \text{block}_0, \text{block}_1, \ldots, \text{block}_{m-2}, \text{block}_{m-1} \]

\[ x[0], x[1], \ldots, x[n-1] \]

\[
\begin{array}{cccc}
    \text{Fetch} & \text{Compute} & \text{Write back} & \\
    \text{dmaget} & \text{dmaget} & \text{dmaget} & \\
\end{array}
\]

while \((i < n/s)\), \(i++\)
DMA Data Transfers: Single Buffering

$s$: number of array elements in one block,

\[
\begin{align*}
\text{block}_0 & \quad \text{block}_1 & \quad \cdots & \quad \text{block}_{m-2} & \quad \text{block}_{m-1} \\
x[0] & \quad x[1] & \quad \cdots & \quad \cdots & \quad x[n-1]
\end{align*}
\]

Identical procedure for each block:

1. Compute(block\(_i\))
2. Fetch(block\(_i\))
3. Compute(block\(_i\))
4. Write back(block\(_i\))

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

\[i++\]
DMA Data Transfers: Single Buffering

$s$: number of array elements in one block,

```
block_0  block_1  block_{m-2}  block_{m-1}
```

\[ x[0] \quad x[1] \quad \ldots \quad x[n-1] \]

Sequential execution of computations and data transfers.

```
i = 0
```

\[ \text{while } (i < n/s) \]

\[ i++ \]

```
Fetch(block_i)
Compute(block_i)
Write back(block_i)
```

\( \text{dma}_\text{put}(\text{block}_i, \text{local-buffer}, s) \)
**DMA Data Transfers: Single Buffering**

$s$: number of array elements in one block,

\[ \begin{array}{ccccccccc}
\text{block}_0 & \text{block}_1 & \text{block}_{m-2} & \text{block}_{m-1} \\
\hline
x[0] & x[1] & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & x[n-1] \\
\end{array} \]

\[ s \quad n \]

\[ i = 0 \]

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

\[ i++ \]

Fetch(block_0)

Compute(block_0)

Write back(block_0): \text{dma_put}(\text{block}_0, \text{local-buffer}, s)

- Sequential execution of computations and data transfers.
DMA Data Transfers: Double Buffering

Asynchronous DMA calls:

\[
dma\_get(local – buffer[1], block_0, s)
\]

\[
dma\_get(local – buffer[2], block_{i+1}, s)
\]

\[
i = 0
\]

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

\[
i++
\]

\[
Compute(block_i)
\]

\[
Write \ back(block_i)
\]

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

\[
Write \ back(block_{(n/s) - 1})
\]

\[
Fetch(block_0)
\]

\[
Fetch(block_{i+1})
\]
DMA Data Transfers: Double Buffering

Asynchronous DMA calls:

- Fetch(block₀)
- Fetch(blockᵢ+1)
- Compute(blockᵢ)
- Write back(blockᵢ)
- Compute(block(n/s)−1)
- Write back(block(n/s)−1)

while (i < (n/s) − 1)

i++

dma_get(local − buffer[1], block₀, s)
dma_get(local − buffer[2], blockᵢ+1, s)
DMA Data Transfers: Double Buffering

Asynchronous DMA calls:

\[
\begin{align*}
&\text{DMA get (local – buffer}[1], \ block_0, \ s) \\
&\text{DMA get (local – buffer}[2], \ block_{i+1}, \ s) \\
&i = 0 \\
&\text{while } (i < (n/s) - 1) \\
&\quad i++ \\
&\text{Compute (block}_i) \\
&\text{Write back (block}_i) \\
&\text{Compute (block}_{n/s} - 1) \\
&\text{Write back (block}_{n/s} - 1)
\end{align*}
\]

- Overlap of computations and data transfers.
DMA Data Transfers: Double Buffering

Asynchronous DMA calls:

- Fetch(block_0)
- dma_get(local − buffer[1], block_0, s)
- i = 0
- Compute(block_i)
- dma_get(local − buffer[2], block_{i+1}, s)
- while (i < (n/s) − 1)
  - i ++
- Write back(block_i)
- Compute(block_{n/s} − 1)
- Write back(block_{n/s}−1)
- Fetch(block_{i+1})

Overlap of computations and data transfers.
Double Buffering Pipelined Execution

Overlap of,

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

Performance can be further improved by an appropriate choice of data granularity.
Double Buffering Pipelined Execution

Overlap of,

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

![Diagram](image_url)

Performance can be further improved by an appropriate choice of data granularity.
Double Buffering Pipelined Execution

Overlap of,

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

Input Transfer

\[
\begin{array}{c}
\text{Input Transfer} \\
\hline
\text{Prologue} \\
\hline
b_0 & b_1 & b_2 \\
\hline
\end{array}
\]

Computation

\[
\begin{array}{c}
\text{Computation} \\
\hline
\text{Prologue} \\
\hline
b_0 & b_1 & b_2 \\
\hline
\end{array}
\]

Output Transfer

\[
\begin{array}{c}
\text{Output Transfer} \\
\hline
\text{Prologue} \\
\hline
b_0 & b_1 & b_2 \\
\hline
\end{array}
\]

Performance can be further improved by an appropriate choice of data granularity.
Double Buffering Pipelined Execution

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

Performance can be further improved by an appropriate choice of data granularity.
Overalp of,

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

Performance can be further improved by an appropriate choice of data granularity.
Double Buffering Pipelined Execution

Overlap of,

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

![Diagram showing overlap of computation and transfer]

Performance can be further improved by an appropriate choice of data granularity.
Granularity of Transfers

**1Dim Data:**
block size $s$

![Diagram showing 1D data blocks and their granularity](image)

**2Dim Data:**
block shape $(s_1, s_2)$

Contribution:
We derive optimal granularity for 1D and 2D DMA transfers,
Granularity of Transfers

- **1Dim Data:** block size $s$

  
  ![Diagram of 1D data]

  

- **2Dim Data:** block shape $(s_1, s_2)$

  
  ![Diagram of 2D data]

Contribution: We derive optimal granularity for 1D and 2D DMA transfers,
Granularity of Transfers

- **1Dim Data:**
  - block size $s$

- **2Dim Data:**
  - block shape $(s_1, s_2)$

Contribution:

We derive optimal granularity for 1D and 2D DMA transfers.
Our Contribution:

We derive **optimal granularity** for 1D and 2D DMA transfers,

- **1Dim** data work was published in *Hipecac 2012*,
  S.Saidi, P.tendulkar, T.Lepley, O.Maler, “Optimizing explicit data transfers for data parallel applicationson the Cell architecture ”

- **2D** data work was published in *DSD 2012*,
  S.Saidi, P.tendulkar, T.Lepley, O.Maler, “Optimal 2D Data Partitioning for DMA Transfers on MPSoCs”
Outline

1. Context and Motivation

2. Contribution
   - Problem Definition
   - Optimal Granularity for a Single Processor
     - 1Dim Data
     - 2Dim Data
   - Multiple Processors
   - Shared Data

3. Experiments on the Cell.BE

4. The move towards Platform 2012

5. Conclusions and Perpectives
Outline

1. Context and Motivation
2. Contribution
   - Problem Definition
   - Optimal Granularity for a Single Processor
     - 1Dim Data
     - 2Dim Data
   - Multiple Processors
   - Shared Data
3. Experiments on the Cell.BE
4. The move towards Platform 2012
5. Conclusions and Perspectives
Outline

1. Context and Motivation

2. Contribution
   - Problem Definition
     - Optimal Granularity for a Single Processor
       - 1Dim Data
       - 2Dim Data
     - Multiple Processors
     - Shared Data

3. Experiments on the Cell.BE

4. The move towards Platform 2012

5. Conclusions and Perspectives
Software Pipelining

- We want to optimize execution of the pipeline.
Software Pipelining

- We want to optimize execution of the pipeline.

Optimal Granularity:
What is the \textbf{Granularity choice} that optimizes performance?
Computation Regime and Transfer Regime

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

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

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

**Proc Idle Time**

**Computation Regime \( C > T \):**
Computation Regime and Transfer Regime

- **T** and **C**: Transfer and Computation time of a block

**Transfer Regime T > C:**

- **Input transfer**
- **Computation**
- **Output transfer**

**Computation Regime C > T:**

- **Input transfer**
- **Computation**
- **Output transfer**
In the Computation Regime:

**Granularity s:**

- **Input transfer:** \( b_0 \), \( b_1 \), \( b_2 \)
- **Computation:** \( b_0 \), \( b_1 \), \( b_2 \)
- **Output transfer:** \( b_0 \), \( b_1 \), \( b_2 \)

**Granularity s': s' > s:**

- **Input transfer:** \( b_0 \), \( b_1 \)
- **Computation:** \( b_0 \), \( b_1 \)
- **Output transfer:** \( b_0 \), \( b_2 \)
Optimal Granularity: Problem Formulation

1Dim Data: block size

Find $s^*$ such that,

$$\text{Min } T(s) \text{ s.t.}$$

$$T(s) \leq C(s)$$

$$(s) \in [1..n]$$

$$s \leq M$$

2Dim Data: block shape

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

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

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

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

$$s_1 \times s_2 \leq M$$
Optimal Granularity: Problem Formulation

1Dim Data: block size

Find $s^*$ such that,

$$\min T(s) \quad \text{s.t.}$$

$$T(s) \leq C(s)$$

$(s) \in [1..n]$ 

$s \leq M$

2Dim Data: block shape

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

$$\min T(s_1, s_2) \quad \text{s.t.}$$

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

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

$s_1 \times s_2 \leq M$
Outline

1. Context and Motivation

2. Contribution
   - Problem Definition
   - Optimal Granularity for a Single Processor
     - 1Dim Data
     - 2Dim Data
   - Multiple Processors
   - Shared Data

3. Experiments on the Cell.BE

4. The move towards Platform 2012

5. Conclusions and Perspectives
Outline

1. Context and Motivation

2. Contribution
   - Problem Definition
   - Optimal Granularity for a Single Processor
     - 1Dim Data
     - 2Dim Data
   - Multiple Processors
   - Shared Data

3. Experiments on the Cell.BE

4. The move towards Platform 2012

5. Conclusions and Perspectives
Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

Computation time $C(s)$:

$$C(s) = \omega \cdot s$$

DMA Transfer time $T(s)$:

$$T(s) = I + \alpha \cdot b \cdot s$$
Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

- \( s \): number of array elements clustered in one (contiguous) block,

Computation time \( C(s) \):

\[
C(s) = \omega \cdot s
\]

DMA Transfer time \( T(s) \):

\[
T(s) = I + \alpha \cdot b \cdot s
\]
Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

- $s$: nb array elements clustered in one (Contiguous) block,

\[
C(s) = \omega \cdot s
\]

\[
T(s) = I + \alpha \cdot b \cdot s
\]

Computation time $C(s)$:

- $\omega$: time to compute one element,
Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

- \( s \): nb array elements clustered in one (Contiguous) block,

\[
C(s) = \omega \cdot s
\]

\[
T(s) = I + \alpha \cdot b \cdot s
\]
Optimal Granularity: 1Dim Data

Characterization of Computation and Transfer Time:

- $s$: nb array elements clustered in one (Contiguous) block,

$$ C(s) = \omega \cdot s $$

$$ T(s) = I + \alpha \cdot b \cdot s $$

- $\omega$: time to compute one element,
- $I$: fixed DMA initialization cost,
Optimal Granularity: 1Dim Data

Characterization of Computation and Transfer Time:

- \( s \): nb array elements clustered in one (Contiguous) block,

Computation time \( C(s) \):

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

\[
C(s) = \omega \cdot s
\]

DMA Transfer time \( T(s) \):

- \( I \): fixed DMA initialization cost,
- \( \alpha \): transfer cost per byte,
Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

- $s$: nb array elements clustered in one (Contiguous) block,

\[ C(s) = \omega \cdot s \]

\[ T(s) = I + \alpha \cdot b \cdot s \]

**Computation time $C(s)$:**
- $\omega$: time to compute one element,

**DMA Transfer time $T(s)$:**
- $I$: fixed DMA initialization cost,
- $\alpha$: transfer cost per byte,
- $b$: size of one array element,
Optimal Granularity : 1Dim Data

Characterization of Computation and Transfer Time:

- \( s \): nb array elements clustered in one (Contiguous) block,

\[
C(s) = \omega \cdot s
\]

\[
T(s) = I + \alpha \cdot b \cdot s
\]
Optimal Granularity : 1Dim Data

Pb Formulation

\[ \text{Min } T(s) \text{ s.t. } \]
\[ T(s) \leq C(s) \]
\[ s \in [1..n] \]
\[ s \leq M \]

- \( s \): block size
- \( M \): Memory limitation

Optimal Granularity \( s^* \):

\[ \begin{align*}
T(s^*) &= C(s^*) \\
T(s) &> C(s) \\
T(s) &< C(s)
\end{align*} \]

Transfer Domain | Computation Domain

Time

Selma Saidi

Optimizing DMA Transfers

24 October 2012
Optimal Granularity: 1Dim Data

Pb Formulation

Min $T(s)$ s.t.

$T(s) \leq C(s)$
$s \in [1..n]$
$s \leq M$

$s$: block size
$M$: Memory limitation

Optimal Granularity $s^*$:

$C(s^*) = T(s^*)$

Computation Domain

Time

$T = C$

$T < C$

$s^*$

$M$

Selma Saidi
Outline

1. Context and Motivation

2. Contribution
   - Problem Definition
   - Optimal Granularity for a Single Processor
     - 1Dim Data
     - 2Dim Data
   - Multiple Processors
   - Shared Data

3. Experiments on the Cell.BE

4. The move towards Platform 2012

5. Conclusions and Perspectives
Characterization of Computation and Transfer Time:

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

Computation time $C(s_1, s_2)$:
Optimal Granularity : 2Dim Data

Characterization of Computation and Transfer Time:

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

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

\(\omega\): time to compute one element,
Optimal Granularity : 2Dim Data

Characterization of Computation and Transfer Time:

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

\[
\text{Computation time } C(s_1, s_2) = \omega \cdot s_1 \cdot s_2
\]
Optimal Granularity: 2Dim Data

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

- $I_1$: transfer cost overhead per line,

\[
T(s_1, s_2) = I + I_1 s_1 + \alpha \cdot b \cdot s_1 \cdot s_2
\]
Optimal Granularity for a Single Processor

Optimal Granularity: 2Dim Data

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

- $l_1$: transfer cost overhead per line,

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

Strided DMA transfers are costlier than contiguous transfers.
Influence of the Block Shape on the DMA Transfer Cost

Different block shapes with same area **BUT** different DMA transfer time,

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

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

\[ (s_1, s_2) = (4, 1) \]
Optimal Granularity : 2Dim Data

- $C(s_1, s_2)$ : computation time of a block,
- $T(s_1, s_2)$ : transfer time of a block,
Optimal Granularity: 2Dim Data

- $C(s_1, s_2)$: computation time of a block,
- $T(s_1, s_2)$: transfer time of a block,
Optimal Granularity: 2Dim Data

\textbf{Pb Formulation}

\begin{align*}
\text{Min } & \quad T(s_1, s_2) \quad \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
\end{align*}

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

\text{Computation Domain}

\text{Transfer Domain}

$T = C$

$T < C$

$T > C$

$s_1$

$s_2$

$l_1 / \psi$

$n_1$

$n_2$
Optimal Granularity : 2Dim Data

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

\[ T(s'_1, s'_2) < T(s_1, s_2) \]

\[ T = C \]
Optimal Granularity : 2Dim Data

\[ T = C \]

\[ n_1 = \frac{1}{2} \left( \frac{1}{\psi} (l_1 + l_0) \right) \]

\[ s_1^* = 1 \]

\[ s_2^* = \frac{1}{\psi} (l_1 + l_0) \]

Optimal granularity is the Contiguous block to reach the computation regime:
Outline

1. Context and Motivation
2. Contribution
   - Problem Definition
   - Optimal Granularity for a Single Processor
     - 1Dim Data
     - 2Dim Data
   - Multiple Processors
     - Shared Data
3. Experiments on the Cell.BE
4. The move towards Platform 2012
5. Conclusions and Perspectives
Partitioning: \( p \) contiguous chunks of data

\[
\begin{array}{cccc}
P_1 & \frac{n}{p} & 1 & \frac{n}{p} \\
\vdots & \vdots & \vdots & \vdots \\
P_2 & 1 & \frac{n}{p} & \frac{n}{p} \\
P_3 & \frac{n}{p} & 1 & \frac{n}{p} \\
P_4 & \frac{n}{p} & 1 & \frac{n}{p} \\
\end{array}
\]

Processors are identical: same local store capacity, same double buffering granularity...etc.
Multiple Processors

Pipelined execution for several processors:

- processors DMA requests are done concurrently,
Multiple Processors

Pipelined execution for several processors:

- Processors DMA requests are done **concurrently**,
Multiple Processors

Pipelined execution for several processors:

- processors DMA requests are done concurrently,
Multiple Processors

Pipelined execution for several processors:

- processors DMA requests are done concurrently,
Multiple Processors

Pipelined execution for several processors:

<table>
<thead>
<tr>
<th>Input Transfer</th>
<th>Prologue</th>
<th>Epilogue</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_0, b_1, b_2$</td>
<td>$b_0, b_1, b_2$</td>
<td>$b_3, b_4, b_5$</td>
</tr>
<tr>
<td>$b_3, b_4, b_5$</td>
<td>$b_3, b_4, b_5$</td>
<td>$b_6, b_7, b_8$</td>
</tr>
<tr>
<td>$b_6, b_7, b_8$</td>
<td>$b_6, b_7, b_8$</td>
<td></td>
</tr>
</tbody>
</table>

Output Transfer

<table>
<thead>
<tr>
<th>Proc Idle time</th>
</tr>
</thead>
</table>

- processors DMA requests are done **concurrently**,

$$T(s, p) = l + \alpha(p) \cdot b \cdot s$$

$\alpha(p)$: transfer cost per byte given contentions of $p$ concurrent transfer requests.
Multiple Processors: Optimal Granularity

- Optimal granularity given \( p \) processors: \( s^*(p) \),

\[
T_1(s) = C(s) \quad \text{(a) One-dimensional data}
\]

\[
T_1(s) = C(n) \quad \text{(b) Two-dimensional data}
\]

\[
s^*(1) = \frac{n}{s^*(1)}
\]
Multiple Processors: Optimal Granularity

- Optimal granularity given \( p \) processors: \( s^*(p) \),

(a) One-dimensional data

(b) Two-dimensional data
Multiple Processors: Optimal Granularity

Optimal granularity given \( p \) processors: \( s^*(p) \),

(a) One-dimensional data

(b) Two-dimensional data

Optimal Granularity **increases** with number of processors
Summary:

We derived optimal granularity,

- **main idea**: balance between computation and transfer time of a block,
- 2D data: **block shape influences transfer time** (overhead per line, $l_1$)
- multiple processors: **number of processors influence transfer time** (with $\alpha(p)$)
Local Memory Constraint

- What if Optimal Granularity does not fit in Local memory?
  1. take available memory space,
  2. reduce the number of processors,

(a) One-dimensional data

(b) Two-dimensional data
Outline

1. Context and Motivation

2. Contribution
   - Problem Definition
   - Optimal Granularity for a Single Processor
     - 1Dim Data
     - 2Dim Data
   - Multiple Processors
   - Shared Data

3. Experiments on the Cell.BE

4. The move towards Platform 2012

5. Conclusions and Perspectives
Applications with shared data: 1Dim

Data parallel loop with shared input data:

for \( i := 0 \) to \( n - 1 \) do
\[
Y[i] := f(X[i], V[i]); \quad V[i] = \{X[i - 1], X[i - 2], ..., X[i - k]\}
\]
od
Applications with shared data: 1Dim

- Data parallel loop with shared input data:

  for $i := 0$ to $n - 1$ do
  $Y[i] := f(X[i], V[i]);$
  $V[i] = \{X[i - 1], X[i - 2], ..., X[i - k]\}$
  od

- Neighboring blocks share data:
Shared Data: 2Dim

- Data parallel loop with shared input data:
  
  for $i := 1$ to $n_1$ do
  for $i := 2$ to $n_2$ do
    $Y[i_1, i_2] := f(X[i_1, i_2], V[i_1, i_2])$;
    $V[i_1, i_2] = \{X[i_1 - 1, i_2], X[i_1, i_2 - 1], ..., X[i_1 - k, i_2]\}$
  od

- symmetric window,

---

Selma Saidi

Optimizing DMA Transfers

24 October 2012
Optimal Granularity: 1Dim Data

Compare strategies for transferring shared data:

1. Replication: via *DMA transfers* from the off-chip memory to local memory.

2. Inter-processor communication: processors exchange data via the *network-on-chip* between the cores;

3. Local buffering: via *local copies* done by the processors.

Based on a parametric study, we derive optimal strategy and granularity for transferring shared data,
we consider Replication for transferring shared data,

$R$: size of replicated data.

$R = 12$

$(s_1, s_2) = (2, 2)$
Optimal Granularity : 2Dim Data

Influence of the block shape on the size of share data:

- Compare Transfer cost of a flat and a square block,

\[ R: \text{size of replicated data.} \]

- \( R = 14 \) \( s_1 = 1 \) \( s_2 = 4 \)

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

- \( R = 12 \) \( s_1 = 2 \) \( s_2 = 2 \)

\( (s_1, s_2) = (2, 2) \)
Optimal Granularity: 2Dim Data

Influence of the block shape on the size of share data:

- Compare Transfer cost of a flat and a square block,

$R$: size of replicated data.

- More Replicated data Overhead

$(s_1, s_2) = (1, 4)$

$(s_1, s_2) = (2, 2)$
Optimal Granularity: 2Dim Data

Influence of the block shape on the size of share data:

- Compare Transfer cost of a flat and a square block,
- $R$: size of replicated data.

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

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

- More Replicated data
  Overhead

- More transfer lines
  Overhead
Optimal Granularity: 2Dim Data

Influence of the block shape on the size of share data:

- Compare Transfer cost of a flat and a square block,

$R$: size of replicated data.

![Diagram](image)

More Replicated data Overhead

More transfer lines Overhead
Optimal Granularity : 2Dim Data

Problem Formulation

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

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

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

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

\[
(s_1 + k) \times (s_2 + k) \leq M
\]
Optimal Granularity : 2Dim Data

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

\[ s_1 = s_2 \]

Selma Saidi

Optimizing DMA Transfers

24 October 2012
Optimal Granularity : 2Dim Data

Optimal shape: between a square and a flat shape,

\[
\begin{align*}
T &= C \
T_k &= C \\
\{ s_1^* &= \Delta + (c_1/\psi)(1/D) \\
\Delta + (l_1/\psi)(1 + D) \\
\}
\end{align*}
\]
Outline

1. Context and Motivation

2. Contribution
   - Problem Definition
   - Optimal Granularity for a Single Processor
     - 1Dim Data
     - 2Dim Data
   - Multiple Processors
   - Shared Data

3. Experiments on the Cell.BE

4. The move towards Platform 2012

5. Conclusions and Perspectives
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>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA issue time</td>
<td>$I$</td>
<td>$\approx 400$ clock cycles</td>
</tr>
<tr>
<td>Off-chip memory transfer cost/byte: 1 proc</td>
<td>$\alpha(1)$</td>
<td>0.22 clock cycles</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>
<tr>
<td>Inter-processor comm transfer cost/byte for</td>
<td>$\beta$</td>
<td>0.13 clock cycles</td>
</tr>
</tbody>
</table>
Optimal Granularity: 1Dim Data, No Sharing

- predicted optimal granularities give good performance.
Optimal Granularity: 1Dim Data, Sharing

Comparing several strategies:

![Diagram showing execution time vs. super-block size for different strategies.]

- 2 repl
- 2 ipc
- 2 local buffering
- 8 repl
- 8 ipc
- 8 local buffering
Optimal Granularity: 2Dim Data, Sharing

- We implement double buffering on a mean filtering algorithm,

- predicted optimal granularities give good performance.
Outline

1. Context and Motivation

2. Contribution
   - Problem Definition
   - Optimal Granularity for a Single Processor
     - 1Dim Data
     - 2Dim Data
   - Multiple Processors
   - Shared Data

3. Experiments on the Cell.BE

4. The move towards Platform 2012

5. Conclusions and Perpectives
P2012 Memory Hierarchy

1. Intra cluster L1 memory (256 Kbytes),
2. Inter cluster L2 memory,
3. Off-chip L3 memory
DMA Latency

we measure the DMA latency on P2012,

DMA performance Model:

\[ T(s, p) = I + \alpha(p)bs \]

\( I = 240 \text{cycles} \)

\begin{tabular}{|c|c|}
\hline
\( p \) & \( \alpha(p) \) \\
\hline
1 & 0.25 \\
2 & 0.45 \\
4 & 0.65 \\
8 & 1.15 \\
16 & 2.15 \\
\hline
\end{tabular}
DMA Latency

we measure the DMA latency on P2012,

DMA performance Model:

\[ T(s, p) = I + \alpha(p)bs \]

\[ I = 240 \text{cycles} \]

<table>
<thead>
<tr>
<th>( p )</th>
<th>( \alpha(p) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>2</td>
<td>0.45</td>
</tr>
<tr>
<td>4</td>
<td>0.65</td>
</tr>
<tr>
<td>8</td>
<td>1.15</td>
</tr>
<tr>
<td>16</td>
<td>2.15</td>
</tr>
</tbody>
</table>
DMA Transfers in a Cluster

Shared Local Memory and shared DMA:
2 approaches for transferring data,

1. **Liberal:** Processors fetch data independently
2. **Collaborative:** Processors fetch data collectively
Liberal Approach: Processors fetch data independently

\[
T(s, p) = l + \alpha(p)bs \\
C(s, p) = o + \omega s \\
s \leq M/p
\]

Program 1 (Liberal): Kernel

```c
Kernel(data_type global *GBuffer, param1, param2 ....)
{
data_type LBuffer [size];
...
async_work_item_copy(GBuffer, LBuffer, size, e);
...
wait_event(e);
}
```

Opencl Kernel:

work item = processor
async_work_item_copy: DMA fetch for each processor
Collab Approach: Processors fetch data Collectively

![Diagram showing the architecture of processors fetching data](image)

\[ T(s, p) = I + \alpha(1)bs \]
\[ C(s, p) = o(p) + (\omega/p)s \]
\[ s \leq M \]

Program 2 (Collaborative): Kernel (data_type global*
GBuffer, data_type local *LBuffer, param1, param2 ,...)
{
...
async_work_group_copy(GBuffer, LBuffer, size, e);
... wait_group_event(e);
}

Opencl Kernel:
work group = cluster
async_work_group_copy: DMA fetch for the cluster
Liberal Approach: Processors fetch data independently

- increase of number of processors reduces max buffer size,
  - double buffering implementation results:
    Optimal Granularity does not fit in the available memory space
Liberal Approach: Processors fetch data independently

- synchronization overhead,

  - double buffering implementation results:
    Performance degradation when increase number of processors
On-going Work:

- find the right balance between number of processors and Memory space budget,
- compare both liberal and collaborative approach,
1. Context and Motivation

2. Contribution
   - Problem Definition
   - Optimal Granularity for a Single Processor
     - 1Dim Data
     - 2Dim Data
   - Multiple Processors
   - Shared Data

3. Experiments on the Cell.BE

4. The move towards Platform 2012

5. Conclusions and Perspectives
Conclusion

We presented a general methodology for automating decisions about Optimal granularity for data transfers,

we capture the following facts,

1. Block shape and size influence the transfer/computation Ratio,
2. DMA performance (sensitivity to the block shapes, number of processors)
3. tradeoff between Strided DMA overhead vs size of replicated data
Perspectives

1. Consider other applications patterns,
2. Capture variations (hardware and Software),
3. Generalize the approach to more than 2 memory levels,
4. Integrate this work in a complete compilation flow,
5. combine task and data parallelism,
6. ...
Thank You !!!