Improving GPU’s energy efficiency by using data regularity

Fabrice Mouhartem

With the team ALF
Outline

1. The GPU power consumption problem
   - Inter-thread regularity

2. Our hardware proposition
   - Nowadays situation
   - Affine vector registers
   - Results

3. Improvement
   - Vertical SIMT
Outline

1. The GPU power consumption problem
   - Inter-thread regularity

2. Our hardware proposition
   - Nowadays situation
   - Affine vector registers
   - Results

3. Improvement
   - Vertical SIMT
## Presentation

### What is a GPU?

A GPU is a unit specialized in graphical operations which is nowadays used for general purpose computing.
Presentation

What is a GPU?
A GPU is a unit specialized in graphical operations which is nowadays used for general purpose computing.

GPU programming overview.

Kernel
- Grids
- Blocks
- Threads

Programming level

Execution level
- Warps (32 threads)
- SIMT execution

SM
The problem

Observation

GPU performance is limited by the energy consumption of the chip.
The problem

Observation
GPU performance is limited by the energy consumption of the chip.

Goal
Improve energy efficiency: lower power for same performances or higher performances.
Improving GPU’s energy efficiency by using data regularity

The GPU power consumption problem

Inter-thread regularity

Observations

Registers are very similar

<table>
<thead>
<tr>
<th>SDMT</th>
<th>12 x 12 y 12 z 12 w</th>
<th>81% of integer registers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 bits 8 bits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDMT</td>
<td>12 x 10 y 12 z 12 w</td>
<td>97% of integer registers.</td>
</tr>
</tbody>
</table>

benchmarks

| BFSKernel | srad_cuda_1 | srad_cuda_2 | bisectKernel | lud_diagonal | lud_internal | lud_perimeter | modulateKernel | BlackScholesGPU | fwtBatch1Kernel | fwtBatch2Kernel | transpose_naive | inverseCNDKernel | bitonicSortShared | histogram64Kernel | bitonicMergeGlobal | bitonicMergeShared | bitonicSortShared1 | histogram256Kernel | needle_cuda_shared_1 | needle_cuda_shared_2 | binomialOptionsKernel | convolutionRowsKernel | bpnn_layerforward_CUDA | mergeHistogram64Kernel | mergeHistogram256Kernel | stencil_3D_16x16_order8 | bpnn_adjust_weights_cuda | convolutionColumnsKernel | MonteCarloOneBlockPerOption | BFSKernel | dwtHaar1D | matrixMul | transpose | Average |

F. Mouhartem

Improving GPU’s energy efficiency by using data regularity

July 11, 2013
And the similar registers are mostly affine.

96% of DDMT registers are affine. 35% of DDMT registers are uniform.
Ideas

Two approaches:
- Compiler level
- Hardware level
Ideas

Two approaches:

- Compiler level
- Hardware level
Ideas

Two approaches:
- Compiler level
- Hardware level

Two solutions:
- Similar registers optimization.
- Affine vector optimization.
Ideas

Two approaches:
- Compiler level
- Hardware level

Two solutions:
- Similar registers optimization.
- Affine vector optimization.
Outline

1. The GPU power consumption problem
   - Inter-thread regularity

2. Our hardware proposition
   - Nowadays situation
   - Affine vector registers
   - Results

3. Improvement
   - Vertical SIMT
Definitions

**Affine vector**

An affine vector is a vector register such that:

\[ v[i] = b + s \times i \]
Definitions

Affine vector

An affine vector is a vector register such that:

\[ v[i] = b + s \cdot i \]

Property

An affine vector is totally defined by its base and its stride.
Definitions

Affine vector
An affine vector is a vector register such that:

\[ v[i] = b + s \times i \]

Property
An affine vector is totally defined by its base and its stride.

Example
\begin{align*}
\text{b} &= 7, \text{s} = 4 \\
7 &\quad 11 &\quad 15 &\quad 19 &\quad 23 &\quad 27 &\quad 31 &\quad 35
\end{align*}
State of the art

Remark

It is possible to predict the nature of the result of an affine operation from nature of the operands.
Improving GPU’s energy efficiency by using data regularity

State of the art

Remark

It is possible to predict the nature of the result of an affine operation from nature of the operands.

\[
\begin{array}{|c|c|c|c|}
\hline
+ & U & A & G \\
\hline
U & U & A & G \\
A & A & A & G \\
G & G & G & G \\
\hline
\end{array}
\quad
\begin{array}{|c|c|c|c|}
\hline
\times & U & A & G \\
\hline
U & U & A & G \\
A & A & G & G \\
G & G & G & G \\
\hline
\end{array}
\quad
\begin{array}{|c|c|c|c|}
\hline
<< & U & A & G \\
\hline
U & U & A & G \\
A & G & G & G \\
G & G & G & G \\
\hline
\end{array}
\]

\[
(b + s \cdot i) + (b' + s' \cdot i) = (b + b') + (s + s') \cdot i
\]

\[
(b + s \cdot i) \times (b' + 0 \cdot i) = (b \cdot b') + (s \cdot b') \cdot i
\]

\[
(b + s \cdot i) \times (b' + s' \cdot i) = b \cdot b' + (s \cdot b' + s' \cdot b) \cdot i + s \cdot s' \cdot i^2
\]
State of the art

We can add an affine tag to vectors [CDZ10].

<table>
<thead>
<tr>
<th>VRF</th>
<th>b, s</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VRF</th>
<th>b, s</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VRF</th>
<th>b, s</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VRF</th>
<th>b, s</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VRF</th>
<th>b, s</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
State of the art

This method does not handle divergences on vector operations.
State of the art

This method does not handle divergences on vector operations.

Divergence

A divergence occurs when an operator does not apply on the entire vector.
State of the art

This method does not handle divergences on vector operations.

**Divergence**

A divergence occurs when an operator does not apply on the entire vector.

**Example of divergence**

\[
\text{mask} = \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 3 & 5 & 7 & 9 \end{bmatrix} + \begin{bmatrix} 5 & 4 & 3 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 8 & 9 & 10 & 9 \end{bmatrix}
\]
Our proposition

- We will lower the granularity of the detection.

```
mask: 0 1 0 0 1 1 1 0
vector: 8 7 2 0 1 5 7 5
```

```
b  s
4  2
```
Our proposition

- We will lower the granularity of the detection.

<table>
<thead>
<tr>
<th>mask</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

\[ b \quad s \]
Our proposition

- We will lower the granularity of the detection.

\[
\begin{align*}
\text{mask} & : 01001110 \\
\text{vector} & : 87201575
\end{align*}
\]
Our proposition

- We will lower the granularity of the detection.

\[
\begin{array}{c}
\text{mask} \quad 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \\
\text{vector} \quad 8 \ 7 \ 2 \ 0 \ 1 \ 5 \ 7 \ 5 \\
\end{array}
\]

\[
\begin{array}{c}
\text{Result vector:} \quad 8 \ 6 \ 2 \ 0 \ 12 \ 14 \ 16 \ 5 \\
\end{array}
\]

Implementation cost:
\[
32 + 32 + 8 \text{ bit per vector (initially } 32 \times 32 = 1024 \text{ bits}) = 7 \%
\]
Our proposition

We will lower the granularity of the detection.

\[
\begin{array}{cccccccc}
mask & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
vector & 8 & 7 & 2 & 0 & 1 & 5 & 7 & 5 \\
\end{array}
\]

\[b \times s + 4 + 6 \times 2\]
Our proposition

- We will lower the granularity of the detection.

**mask**

| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |

| vector | 8 | 7 | 2 | 0 | 1 | 5 | 7 | 5 |

\[ b \times s \]

**Result vector:**

\[ 4 + 6 \times 2 \]

**Implementation cost**

\[ 32 + 32 + 8 \text{ bit per vector (initially } 32 \times 32 = 1024 \text{ bits}) = 7\% \]
Our proposition

- We will lower the granularity of the detection.

```
mask: 0 1 0 0 1 1 1 0
vector: 8 7 2 0 1 5 7 5
```

Result vector: 8 6 2 0 12 14 16 5

$$4 + 6 \times 2$$
Our proposition

- We will lower the granularity of the detection.

\[
\begin{array}{ccccccc}
\text{mask} & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
\text{vector} & 8 & 7 & 2 & 0 & 1 & 5 & 7 & 5 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
\text{Result vector} & 8 & 6 & 2 & 0 & 12 & 14 & 16 & 5 \\
\end{array}
\]

Implementation cost

\[32 + 32 + 8 \text{ bit per vector (initially } 32 \cdot 32 = 1024 \text{ bits}) = 7\%\]
Our proposition

- Execution pipeline:

  IF → ID → Scalar execution (int) → WB

  WB

  pipe line depth: 8

  Vector execution (FP) → WB
Our proposition

Execution pipeline:

IF → ID → Scalar execution (int) → WB

Vector execution (FP) → WB

pipeline depth: 8
Our proposition

Problem

How can we handle hybrid operations?
Our proposition

Problem
How can we handle hybrid operations?

Flatten
We add a flatten operation to turn partial affine vectors into generic vectors.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3</th>
<th>1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>affine</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten</td>
</tr>
<tr>
<td>generic</td>
</tr>
</tbody>
</table>

F. Mouhartem
Improving GPU’s energy efficiency by using data regularity
July 11, 2013 16/30
82% average hits.
Results

- **Speedup**: 16%
Results

- Speedup: 16%
Results

- Speedup: 16%

![Bar chart showing speedup and slowdowns with overhead of flatten]
Results

- General speedup.
Results

- General speedup.
- Simple to implement.
  - Add of the scalar register file.
  - Implementation of the flatten operation.
Results

- General speedup.
- Simple to implement.
  - Add of the scalar register file.
  - Implementation of the flatten operation.
- Use of a scalar functional unit: lower energy use.
Results

- General speedup.
- Simple to implement.
  - Add of the scalar register file.
  - Implementation of the flatten operation.
- Use of a scalar functional unit: lower energy use.
- Slowdowns can happen.
Results

- General speedup.
- Simple to implement.
  - Add of the scalar register file.
  - Implementation of the flatten operation.
- Use of a scalar functional unit: lower energy use.
- Slowdowns can happen.
  - Reason: flatten at the beginning.
  - We will try to avoid it.
Outline

1. The GPU power consumption problem
   - Inter-thread regularity

2. Our hardware proposition
   - Nowadays situation
   - Affine vector registers
   - Results

3. Improvement
   - Vertical SIMT
Observation

35% of flattens come from generic operations.
Vertical SIMT: State of the art [Kra13]
Vertical SIMT : State of the art [Kra13]
Vertical SIMT: State of the art [Kra13]
Vertical SIMT: State of the art [Kra13]
Improving GPU's energy efficiency by using data regularity

**Vertical SIMT**

**Vertical SIMT : our proposal**

- **RF**
- **SRF**
- **VRF**

- Variables: \( v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7 \)

- Operations: generic, affine, EX, s, +s, time
Improving GPU’s energy efficiency by using data regularity

Vertical SIMT: our proposal

RF

SRF   VRF

$V_0$  $V_1$  $V_2$  $V_3$  $V_4$  $V_5$  $V_6$  $V_7$

generic

affine

$b$ $s$

EX

+s

time
Vertical SIMT: our proposal
Vertical SIMT

Results:
- No affine information loss.
- No flatten execution.
10% more speedup compared to our previous solution.
Vertical SIMT

- 29% baseline speedup.
Results

- No more slowdowns.
- Higher speedup.
Results

- No more slowdowns.
- Higher speedup.
- We can use a separate register file: no more useless vector read.
Conclusion

- We proposed an architecture which decreases the energy consumption of GPU by taking advantage of inter-thread value correlation.
- Other possible ways: taking advantage of the SDMT structure to compress data.
- We could also use hints from the compiler to add an instruction word to allow software optimizations.
- Current work: combine with affine vector cache [CK+11].
Thank you very much for your attention.


More implementable pipeline

![Pipeline Diagram]

**Scalor execution (int)**

**Vector execution (FP)**

**Pipeline depth: 8**

WB

---

F. Mouhartem

Improving GPU's energy efficiency by using data regularity

July 11, 2013
The replay problem

Situation

Horizontal SIMT, we have a flatten operation to handle conflicts.

Problem

If there are affine operands in a generic operation, the affine vectors has to be flattened, and then the operation replayed in a generic fashion.

- Origins:
  - Hybrid operations.
  - Overflow from an affine operation.

- Costs:
  - The operation is twice longer in the pipeline (flatten + operation).
  - We lose the affine information in a read.