On-the-Fly Elimination of Dynamic Irregularities for GPU Computing

Eddy Z. Zhang, Yunlian Jiang, Ziyu Guo, Kai Tian, and Xipeng Shen

The College of William and Mary
Graphic Processing Units (GPU)
Graphic Processing Units (GPU)
Graphic Processing Unit (GPU)

- Massive parallelism
- Favorable
  - computing power
  - cost effectiveness
  - energy efficiency
Graphic Processing Unit (GPU)

- Massive parallelism
- Favorable
  - computing power
  - cost effectiveness
  - energy efficiency
Great for regular computation!

- Massive parallelism
- Favorable
  - computing power
  - cost effectiveness
  - energy efficiency
Graphic Processing Unit (GPU)

Great for regular computation!

A SIMD group (warp)

- Massive parallelism
- Favorable
  - computing power
  - cost effectiveness
  - energy efficiency
Dynamic Irregularities

memory

\[ ... = A[P[tid]]; \]

\[ P[] = \{0, 1, 2, 3, 4, 5, 6, 7\} \]

\[ \text{tid: } \begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array} \]

A[] (a mem seg.)

Dynamic Irregularities

memory

... = A[P[tid]];

\[ \text{tid: } 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \]

\[ \text{P[ ] = \{ 0, 1, 2, 3, 4, 5, 6, 7 \}} \]

\[ \text{A[ ]: } \]

\[ \text{a mem seg.} \]
Dynamic Irregularities

memory

\[ ... = A[P[tid]] ; \]

\[ P[] = \{ 0, 1, 2, 3, 4, 5, 6, 7 \} \]

\[ tid: \begin{array}{cccccccc} 
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 
\end{array} \]

\[ A[]: \]

a mem seg.
Dynamic Irregularities

memory

\[ A[ ] = \{ 0, 5, 1, 7, 4, 3, 6, 2 \} \]

... = \( A[P[tid]] \);

A[]:

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

P[] = \{ 0, 5, 1, 7, 4, 3, 6, 2 \}

tid: 0 1 2 3 4 5 6 7

a mem seg.
Dynamic Irregularities

memory

\[ P[\ ] = \{0, 5, 1, 7, 4, 3, 6, 2\} \]

... = \( A[P[tid]] \);

a mem seg.
Dynamic Irregularities

memory

... = A[P[tid]];

P[] = {0, 5, 1, 7, 4, 3, 6, 2}

tid: 0 1 2 3 4 5 6 7

A[]:

a mem seg.
Dynamic Irregularities

memory

A[P[tid]];  

control flow (thread divergence)

... = A[P[tid]];  

if (A[tid]) { ... }  

A[]:  

0 0 6 0 0 2 4 1  

tid:  

0 1 2 3 4 5 6 7  

A[]:  

0 0 6 0 0 2 4 1  

tid:  

0 1 2 3 4 5 6 7  

a mem seg.
Dynamic Irregularities

memory

A[ ]: [0, 6, 0, 2, 4, 1]

control flow (thread divergence)

if (A[tid]) {...}

Memory segments: A[ ] = {0, 6, 0, 2, 4, 1}
Dynamic Irregularities

memory

```
P[ ] = { 0, 5, 1, 7, 4, 3, 6, 2}
```

```
... = A[P[tid]];  
```

```
tid:  0   1  2   3  4   5   6   7
```

```
A[ ]: |   |   |   |   |   |   |   |
```

a mem seg.

control flow (thread divergence)

```
tid:  0  1  2  3  4  5  6  7
```

```
A[ ]: 0 0 6 0 2 4 1
```

```
if (A[tid]) {...}
```

```
for (i=0; i<A[tid]; i++) {...}
```
Dynamic Irregularities

memory

A[]: tid: 0 1 2 3 4 5 6 7
... = A[P[tid]]; 

control flow (thread divergence)

for (i=0; i<A[tid]; i++) {...}

A[]: tid: 0 1 2 3 4 5 6 7

if (A[tid]) {...}

Degrade throughput by up to \((warp \ size - 1)\) times.

(warp size = 32 in modern GPUs)
Performance Impact

- Applications: Dynamic programming, fluid simulation, image reconstruction, data mining, ...

Potential Speedup

<table>
<thead>
<tr>
<th>Application</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMMER</td>
<td>5.27</td>
</tr>
<tr>
<td>3D-LBM</td>
<td>1.46</td>
</tr>
<tr>
<td>CUDA-EC</td>
<td>1.5</td>
</tr>
<tr>
<td>NN</td>
<td>2.51</td>
</tr>
<tr>
<td>CFD</td>
<td>2.75</td>
</tr>
<tr>
<td>CG</td>
<td>1.8</td>
</tr>
<tr>
<td>Unwrap</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Host: Xeon 5540.
Device: Tesla 1060.
Previous Work

• Mostly on Static Memory Irregularities
  ❖ [Baskaran+, ICS’08], [Lee+, PPoPP’09], [Yang+, PLDI’10], etc.

• Dynamic Irregularity
  ❖ Remain unknown until runtime
  ❖ Mostly through hardware extensions.
    ❖ [Meng+, ISCA’10], [Tarjan+, SC’09], [Fung+, MICRO’07], etc.
Open Questions for Dyn. Irreg. Removal

• Is it possible to do so w/o hw ext.?
Open Questions for Dyn. Irreg. Removal

• Is it possible to do so w/o hw ext.?

• More fundamentally
  • Relations among data, threads, & irregularities?
  • What layout or thread-data mappings minimize the irreg.? How to find them? Complexity?
  • How to resolve conflicts among irregularities? Dependences?
Overview of this Work

- Analytic findings on properties of dyn. irreg. removal

- A software solution: G-Streamline library
  - No profiling or hw ext.
  - Transparent removal on the fly
  - Jeopardize no basic efficiency
  - Treat both types of irreg. holistically
Basic Insight:
Both mem & control irreg. stem from inferior thread-data mappings.

A unified treatment.
Basic Insight:
Both mem & control irreg. stem from inferior thread-data mappings.

A unified treatment.

Two basic mechanisms:
• data relocation
• reference redirection
  • $A[p[tid]] \rightarrow A[q[tid]]$

Compose three transformations.
Trans-1: Data Reordering
(for mem irreg only)

original

\[ \ldots = A[P[tid]]; \]

\[ P[\ ] = \{0,5,2,3,2,3,7,6\} \]

\[ A[\ ]:\]

\[ \text{tid: } 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \]

\[ \text{\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow} \]

\[ \text{A[\ ]:} \]

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{tid: thread ID; } \downarrow : \text{a thread; } \downarrow : \text{data access; } \downarrow : \text{data swapping} \]
Trans-1: Data Reordering
(for mem irreg only)

original

... = A[P[tid]];
P[ ] = {0,5,2,3,2,3,7,6}

A[ ]:                   A'[ ]:

<relocation>

tid: thread ID; § : a thread; ↓ : data access; ↓ : data swapping
Trans-1: Data Reordering
(for mem irreg only)

original

... = A[P[tid]];
P[ ] = {0, 5, 2, 3, 2, 3, 7, 6}

A[ ]:

<relocation>

A'[ ]:

tid: 0 1 2 3 4 5 6 7

tid: thread ID;  : a thread;  : data access;  : data swapping
Trans-1: Data Reordering
(for mem irreg only)

\[ P[\ ] = \{0,5,2,3,2,3,7,6\} \]

\[ \ldots = A[P[tid]]; \]

Maintain mapping between threads & data values
Trans-1: Data Reordering
(for mem irreg only)

original

... = A[P[tid]];  

\[ P[ ] = \{0,5,2,3,2,3,7,6\} \]

maintain mapping between threads & data values
Trans-1: Data Reordering
(for mem irreg only)

original

... = A[P[tid]];  

\[ P[ ] = \{0,5,2,3,2,3,7,6\} \]

maintain mapping between threads & data values

\[ \text{tid: thread ID; } \downarrow \text{: a thread; } \downarrow \text{: data access; } \downarrow \text{: data swapping} \]
Trans-1: Data Reordering
(for mem irreg only)

original

... = A[P[tid]];  

P[ ] = {0, 5, 2, 3, 2, 3, 7, 6}

<redirection>

maintain mapping between threads & data values

transformed

... = A'[Q[tid]];  

Q[ ] = {0, 1, 2, 3, 2, 3, 6, 7}

<relocation>


<table>
<thead>
<tr>
<th>tid: thread ID;</th>
<th>: a thread;</th>
<th>: data access;</th>
<th>: data swapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>↓</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The College of William and Mary

eddy@cs.wm.edu
Trans-2: Job Swapping (for mem)

- Job = operations + data elements accessed

```
original

... = A[P[tid]];  
P[ ] = {0,5,2,3,2,3,7,6}
```

```
A[ ]:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

tid:

```
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
```

```
Trans-2: Job Swapping (for mem)

- Job = operations + data elements accessed

original

\[ \ldots = A[P[tid]]; \]

\[ P[] = \{0,5,2,3,2,3,7,6\} \]
Trans-2: Job Swapping (for mem)

- Job = operations + data elements accessed

**original**

```plaintext
... = A[P[tid]];

P[ ] = {0,5,2,3,2,3,7,6}
```

```
    0 1 2 3 4 5 6 7
original
```

```
    0 1 2 3 4 5 6 7
A[ ]:
```

```
    0 1 2 3 4 5 6 7
tid:
```

```
    0 1 2 3 4 5 6 7
tid:
```
Trans-2: Job Swapping (for mem)

- Job = operations + data elements accessed

original

\[ \ldots = A[P[tid]]; \]

\[ P[ ] = \{0, 5, 2, 3, 2, 3, 7, 6\} \]

A[ ]:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

tid: 0 1 2 3 4 5 6 7

\[ tid: 0 1 2 3 4 5 6 7 \]

A[ ]:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A[ ]:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

eddy@cs.wm.edu
Trans-2: Job Swapping (for mem)

- Job = operations + data elements accessed

original

```plaintext
... = A[P[tid]];
P[ ] = {0,5,2,3,2,3,7,6}
```

```
tid: 0 1 2 3 4 5 6 7
A[ ]:
```

```
tid: 0 1 2 3 4 5 6 7
A[ ]:
```

```
tid: 0 1 2 3 4 5 6 7
A[ ]:
```

```
tid: 0 1 2 3 4 5 6 7
A[ ]:
```
Trans-2: Job Swapping (for mem)

- Job = operations + data elements accessed

original

... = A[P[tid]];

P[ ] = {0, 5, 2, 3, 2, 3, 7, 6}

<redirection>

transformed

newtid = Q[tid];

... = A[P[newtid]];

Q[ ] = {0, 4, 2, 3, 1, 5, 6, 7}
Trans-2: Job Swapping (for mem)

- Job = operations + data elements accessed

Both reduce 1 mem trans.

1 more than the optimal....

Newtid = Q[tid];

\[ \cdots = A[P[newtid]] \]

Q[ ] = \{0,4,2,3,1,5,6,7\}

<redirection>

Original

\[ \cdots = A[P[tid]] \]

P[ ] = \{0,5,2,3,2,3,7,6\}

Transformed

\[ \cdots = A[P[tid]] \]

tid: 0 1 2 3 4 5 6 7

A[ ]:

Both reduce 1 mem trans.

1 more than the optimal....

tid: 0 1 2 3 4 5 6 7

A[ ]:
Trans-2: Job Swapping (for control)

original

if (B[tid]) {...}
Trans-2: Job Swapping (for control)

original

if (B[tid]) {...}

B[ ]: 0 0 6 0 0 2 4 1

tid: 0 1 2 3 4 5 6 7

tid: 0 1 2 3 4 5 6 7
Trans-2: Job Swapping (for control)

original

if (B[tid]) {...}
Trans-2: Job Swapping (for control)

```
if (B[tid]) {...}
```

Original:

```
B[]: 0 0 6 0 0 2 4 1
tid: 0 1 2 3 4 5 6 7
```

Modified:

```
B[]: 0 0 6 0 0 2 4 1
tid: 0 1 2 3 4 5 6 7
```
Trans-2: Job Swapping (for control)

```plaintext
original

if (B[tid]) {...}
```
Trans-2: Job Swapping (for control)

original

if (B[tid]) {...}

<redirection>

transformed

newtid = D[tid];
if (B[newtid]) {...}

D[ ] = {0,1,4,3,2,5,6,7}
Trans-2: Job Swapping (for control)

original

```
if (B[tid]) {...}
```

<redirection>

transformed

```
newtid = D[tid];
if (B[newtid]) {...}
```

\[ D[\ ] = \{0, 1, 4, 3, 2, 5, 6, 7\} \]

Mem ref. pattern changes.
Trans-2: Job Swapping (for control)

B[ ]: 0 0 6 0 0 2 4 1

D[ ] = {0,1,4,3,2,5,6,7}

newtid = D[tid];
if (B[newtid]) {...}

<redirection>

Mem ref. pattern changes.

Solution: A follow-up data reordering.
Trans-2: Job Swapping (for control)

method 1

original

if (B[tid]) {...}

<redirection>

transformed

newtid = D[tid];
if (B[newtid]) {...}

D[ ] = {0,1,4,3,2,5,6,7}

B[ ]:

0 0 6 0 0 2 4 1

Mem ref. pattern changes.
Solution: A follow-up data reordering.
Trans-2: Job Swapping (for control)

original

if (B[tid]) {...}

method 2
Trans-2: Job Swapping (for control)

original

method 2

```c
if (B[tid]) {...}
```

```
B[ ]: [0 0 6 0 0 2 4 1]
```

<relocation>

```
tid: [0 1 2 3 4 5 6 7]
```

```
B'[ ]: [0 0 0 0 6 2 4 1]
```
Trans-2: Job Swapping (for control)

Method 2

Original

if (B[tid]) {...}

Transformed

<relocation>

B[tid]: 0 0 6 0 0 2 4 1
B'[tid]: 0 0 0 0 6 2 4 1
Trans-2: Job Swapping (for control)

original

if (B[tid]) {...}

transformed

<relocation>

if (B'[tid]) {...}

B[ ]:

B'[ ]:

tid:

0 1 2 3 4 5 6 7

0 0 6 0 0 2 4 1

0 0 0 0 6 2 4 1

0 1 2 3 4 5 6 7
Trans-2: Job Swapping (for control)

- **method 2**
- **Job integrity**
  - if (B[tid]) {...}  
  - if (B'[tid]) {...}

### Original Version

```c
if (B[tid]) {...}
```

### Transformed Version

```c
if (B'[tid]) {...}
```

**<relocation>**

<table>
<thead>
<tr>
<th>tid:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>B[]</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>B'[ ]</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>tid:</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Trans-2: Job Swapping (for control)

**Method 2**

Original:

```c
if (B[tid]) {...}
```

Transformed:

```c
if (B'[tid]) {...}
```

### Job Integrity

```c
if (B[tid] - tid) {...}
```

newtid = Q[tid];

```c
if (B'[tid] - newtid) {...}
```

Q[] = {0,1,4,3,2,5,6,7}

---

**Example**: 

<table>
<thead>
<tr>
<th>tid</th>
<th>B[]</th>
<th>B'[ ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
<td>0 0 6 0 0 2 4 1</td>
<td>0 0 0 0 6 2 4 1</td>
</tr>
</tbody>
</table>

<relocation>
Trans-3: Hybrid

- Job swap + Data reorder
- Data reorder + Job swap
Trans-3: Hybrid

- Job swap + Data reorder
- Data reorder + Job swap

\[ P\left[ \right] = \{0,5,2,3,2,3,7,6\} \]

A\left[ \right]:

(original)

\[
... = A[P[tid]];
\]

tid: \[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array}
\]

A\left[ \right]:
Trans-3: Hybrid

- Job swap + Data reorder
- Data reorder + Job swap

Single opt reduces 1 mem trans.
1 more than optimal.
Trans-3: Hybrid

- Job swap + Data reorder
- Data reorder + Job swap

Single opt reduces 1 mem trans.
1 more than optimal.
Trans-3: Hybrid

- Job swap + Data reorder
- Data reorder + Job swap

Single opt reduces 1 mem trans.
1 more than optimal.

original

\[ ... = A[P[tid]]; \]
\[ P[ ] = \{0,5,2,3,2,3,7,6\} \]

\[ ... = A'[Q[tid]]; \]
\[ Q[ ] = \{4,5,2,3,2,3,7,6\} \]

\[ tid: \]
\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \]

A[ ]:

A'[ ]:

<redirection>

<relocation>

P[ ] = \{0,5,2,3,2,3,7,6\}

0   1  2   3  4   5   6   7

original data

reordering

= A'[Q[tid]];

= A[P[tid]];

The College of William and Mary
Trans-3: Hybrid

- Job swap + Data reorder
- Data reorder + Job swap

Single opt reduces 1 mem trans. 1 more than optimal.

Original

\[ ... = A'[Q[tid]]; \]
\[ P[ ] = \{0,5,2,3,2,3,7,6\} \]
\[ A[ ] = \{0,5,2,3,2,3,7,6\} \]
\[ \text{tid: } 0 1 2 3 4 5 6 7 \]

Data reorder

\[ ... = A'[Q[tid]]; \]
\[ Q[ ] = \{4,5,2,3,2,3,7,6\} \]
\[ A'[ ] = \{0,5,2,3,2,3,7,6\} \]
\[ \text{tid: } 0 1 2 3 4 5 6 7 \]

Job swap
Trans-3: Hybrid

- Job swap + Data reorder
- Data reorder + Job swap

Single opt reduces 1 mem trans. 1 more than optimal.
Trans-3: Hybrid

- Job swap + Data reorder
- Data reorder + Job swap

Single opt reduces 1 mem trans.  1 more than optimal.
Trans-3: Hybrid

- Job swap + Data reorder
- Data reorder + Job swap

Single opt reduces 1 mem trans.
1 more than optimal.
Trans-3: Hybrid

- Job swap + Data reordering
- Data reorder + Job swap

Single opt reduces 1 mem trans. 1 more than optimal.

original

\[ ... = A[P[tid]]; \]

\[ P[ ] = \{0,5,2,3,2,3,7,6\} \]

\[ Q[ ] = \{4,5,2,3,2,3,7,6\} \]

transformed

\[ ntid = R[tid]; \]

\[ ... = A'[Q[ntid]]; \]

\[ R[ ] = \{4,5,2,3,0,1,6,7\} \]
Trans-3: Hybrid

- Job swap + Data reorder
- Data reorder + Job swap

Single opt reduces 1 mem trans.

1 more than optimal.
Trans-3: Hybrid

- Job swap + Data reorder
- Data reorder + Job swap

Optimal achieved.
Comparisons

• Irreg. Mem
  • Diff. applicability of reordering and job swapping
  • Hybrid: largest potential

• Irreg. Control
  • Job swapping by redirection
    • lower overhead, but with side effects
  • Job swapping by relocation
    • higher overhead, no side effects
Efficiency control

Adaptive CPU-GPU pipelining

Three-level efficiency-driven adaptation

guidance for transformations

Optimality & approximation

Complexity analysis

Approximating optimal layouts and mappings

How to determine optimal layouts / thread-data mapping?
• NP-Complete
  • Layout: 3D matching
  • Mapping: Partition Problem
• Approx.
  • Duplication/packing
  • Clustered sorting

How to determine optimal layouts / thread-data mapping?
After Transformation

- Benchmark Suites: Rodinia, Tesla Bio, and etc.
- Host: Xeon 5540. Device: Tesla 1060
After Transformation

- Benchmark Suites: Rodinia, Tesla Bio, and etc.
- Host: Xeon 5540. Device: Tesla 1060

<table>
<thead>
<tr>
<th>Benchmark Suite</th>
<th>Without Overhead</th>
<th>With Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMMER</td>
<td>5.27</td>
<td>2.49</td>
</tr>
<tr>
<td>3D-LBM</td>
<td>1.46</td>
<td>1.5</td>
</tr>
<tr>
<td>CUDA-EC</td>
<td>1.1</td>
<td>0.45</td>
</tr>
<tr>
<td>NN</td>
<td>2.51</td>
<td>1.6</td>
</tr>
<tr>
<td>CFD</td>
<td>1.6</td>
<td>0.38</td>
</tr>
<tr>
<td>CG</td>
<td>1.8</td>
<td>0.3</td>
</tr>
<tr>
<td>Unwrap</td>
<td>3.6</td>
<td>0.27</td>
</tr>
</tbody>
</table>
After Transformation

- Benchmark Suites: Rodinia, Tesla Bio, and etc.
- Host: Xeon 5540. Device: Tesla 1060

How to minimize or hide overhead?
G-Streamline

Efficiency control
Adaptive CPU-GPU pipelining
Three-level efficiency-driven adaptation

❖ Transparent, on-the-fly
❖ Adaptive to pattern changes
❖ No perf. degradation
❖ Resilient to dependence
❖ Automatically balance benefits and overhead
- CPU-GPU pipelining
- Kernel splitting
- Partial transf. and overlap.
- Two-level adaptive control

- Transparent, on-the-fly
- No perf. degradation
- Automatically balance benefits and overhead

- Adaptive to pattern changes
- Resilient to dependence

G-Streamline

overhead hiding & minimization

Efficiency control

Adaptive CPU-GPU pipelining
Three-level efficiency-driven adaptation
• CPU-GPU pipelining
• Kernel splitting
• Partial transf. and overlap.
• Two-level adaptive control

❖ Transparent, on-the-fly
❖ Adaptive to pattern changes
❖ No perf. degradation
❖ Resilient to dependence
❖ Automatically balance benefits and overhead
CPU-GPU Pipelining

- Utilize Idle CPU Time
  - Transform on CPU while computing on GPU
  - Automatic shutdown when necessary

```matlab
def for i=1:n
    async_transform (i+2);
    async_copy (i+2);
    gpu_kernel(i);
end
```

```matlab
  cpu_transform( )
  copy_to_gpu
  gpu_kernel
```
Dependence or No Loop

**CFD**
(grid Euler solver)

```matlab
for i=1:iterations
    ...
    cuda_compute_flux(...); // write A, read B
    cuda_time_step(...); // read A, write B
    ...
end
```
Dependence or No Loop

CFD (grid Euler solver)

for i = 1:iterations
    ...
    cuda_compute_flux(...); // write A, read B
    cuda_time_step(...); // read A, write B
    ...
end

CUDA-EC (DNA error correction)

main (){
    ...
    gpu_fix_errors1();
    ...
}
Kernel Splitting

- Also useful for loops with no dependences
  - Enable partial transformation

```c
gpuKernel_org<<<...>>>(pData,...);

gpuKernel_org_sub<<<...>>>(pData,0, (1-r)*len, ...);

pipeline

gpuKernel_opt_sub<<<...>>>(pData,(1-r)*len+1, len, ...);
```
- CPU-GPU pipelining
- Kernel splitting
- Partial transf. and overlap.
- Two-level adaptive control

- Transparent, on-the-fly
- No perf. degradation
- Automatically balance benefits and overhead
- Adaptive to pattern changes
- Resilient to dependence
Final Speedup

- **HMMER**: 2.5
- **3D-LBM**: 1.4
- **CUDA-EC**: 1.1
- **NN**: 0.45
- **CFD**: 0.38
- **CG**: 0.3
- **Unwrap**: 0.27

Legend:
- Basic transformation
- w/ efficiency control
- full potential
Final Speedup

<table>
<thead>
<tr>
<th></th>
<th>Basic transformation</th>
<th>Basic transformation w/ efficiency control</th>
<th>full potential</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMMER</td>
<td>2.5</td>
<td>2.5</td>
<td>5.27</td>
</tr>
<tr>
<td>3D-LBM</td>
<td>1.4</td>
<td>1.4</td>
<td>4.66</td>
</tr>
<tr>
<td>CUDA-EC</td>
<td>1.1</td>
<td>1.2</td>
<td>1.5</td>
</tr>
<tr>
<td>NN</td>
<td>0.45</td>
<td>1.8</td>
<td>2.51</td>
</tr>
<tr>
<td>CFD</td>
<td>0.45</td>
<td>0.38</td>
<td>2.75</td>
</tr>
<tr>
<td>CG</td>
<td>0.3</td>
<td>1.8</td>
<td>1.8</td>
</tr>
<tr>
<td>Unwrap</td>
<td>0.27</td>
<td>2.08</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Speedup
# HW Profiling Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Irreg. Type</th>
<th>Memory Transactions Reduction</th>
<th>Divergent Branch Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D-LBM</td>
<td>div &amp; mem</td>
<td>10%</td>
<td>99%</td>
</tr>
<tr>
<td>CFD</td>
<td>mem</td>
<td>12%</td>
<td></td>
</tr>
<tr>
<td>CG</td>
<td>mem</td>
<td>19%</td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td>mem</td>
<td>67%</td>
<td></td>
</tr>
<tr>
<td>Unwrap</td>
<td>mem</td>
<td>19%</td>
<td></td>
</tr>
<tr>
<td>CUDA-EC</td>
<td>div</td>
<td></td>
<td>10%</td>
</tr>
<tr>
<td>GPU-HMMMER</td>
<td>div</td>
<td></td>
<td>99%</td>
</tr>
</tbody>
</table>
Conclusions

- Comprehensive Understanding
  - Computational complexity
  - Relations among threads, data, and irregularities

- **G-Streamline**: A Unified, Software Solution
  - Treat both mem. and control irregularities
  - Need no offline prof. or hw ext.
  - A whole-system synergy

---

The College of William and Mary

eddy@cs.wm.edu
Thanks!
Adaptive Efficiency Control

*Automatic Shutdown*

- Early Termination if Necessary
- Monitor + Scheduler
  - Whole-system communication and status track
    - GPU status through cudaStreamQuery()
  - Coordination of CPU, GPU, and data transfer
Adaptive Efficiency Control

*Automatic Shutdown*

- Early Termination if Necessary
- Monitor + Scheduler
  - Whole-system communication and status track
  - GPU status through `cudaStreamQuery()`
  - Coordination of CPU, GPU, and data transfer
Adaptive Efficiency Control

*Automatic Shutdown*

- Early Termination if Necessary
- Monitor + Scheduler
  - Whole-system communication and status track
  - GPU status through cudaStreamQuery()
  - Coordination of CPU, GPU, and data transfer
Adaptive Efficiency Control

Fine-Grained Adaptation

• To Determine a Good Splitting Ratio
  ❖ 1. Transparent on-line profiling in first several iterations
  ❖ 2. Build linear model \( \Rightarrow \) Opt. Ratio = \( R_{opt} \)
  ❖ 3. Continue adjusting \( R_{opt} \) through a runtime control system
    ❖ Fast converge to near optimal
    ❖ Avoid unnecessary oscillations
    ❖ Avoid being misled by early shutdown

See Paper Please.