Unleashing the Power of Multi-GPU Accelerators with FLAME

Enrique S. Quintana Ortí
Universidad Jaume I
quintana@icc.uji.es

SIAM Conference on Parallel Processing for Scientific Computing
Seattle, 2010
ACKNOWLEDGMENTS

- Joint work:
  - UJI
    - Francisco D. Igual
    - Mercedes Marqués
    - Gregorio Quintana
  - UT@Austin
    - Ernie Chan
    - Robert van de Geijn
    - Field G. Van Zee
ACKNOWLEDGMENTS

- Support:
  UJI
  UT@Austin

2008 NVIDIA Professorship Award

ClearSpeed™
MOTIVATION

MYTH OR REALITY?

Peak GFLOPs/s

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NV30</td>
<td>NV35</td>
<td>NV40</td>
<td>G70</td>
<td>G71</td>
<td>G80</td>
<td>G80 Ultra</td>
<td>G92</td>
</tr>
<tr>
<td>3.0 GHz</td>
<td>Core2 Duo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.2 GHz</td>
<td>Harpertown</td>
<td></td>
</tr>
</tbody>
</table>

GT200 = GeForce GTX 280  
G71 = GeForce 7900 GTX  
G92 = GeForce 9800 GTX  
G80 = GeForce 8800 GTX  
NV35 = GeForce FX 5950 Ultra  
NV30 = GeForce FX 5800  
NV40 = GeForce 6800 Ultra
MOTIVATION

PERSON PER MONTH?

Peak GFLOP/s

<table>
<thead>
<tr>
<th>NVIDIA GPU</th>
<th>Intel CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT200</td>
<td>GT200</td>
</tr>
<tr>
<td>G92</td>
<td>G92</td>
</tr>
<tr>
<td>G80 Ultra</td>
<td>G80 Ultra</td>
</tr>
<tr>
<td>G80</td>
<td>G80</td>
</tr>
<tr>
<td>G71</td>
<td>G71</td>
</tr>
<tr>
<td>3.0 GHz Core2 Duo</td>
<td>3.2 GHz Harpertown</td>
</tr>
<tr>
<td>NV35</td>
<td>NV35</td>
</tr>
<tr>
<td>NV40</td>
<td>NV40</td>
</tr>
<tr>
<td>NV30</td>
<td>NV30</td>
</tr>
<tr>
<td>G70</td>
<td>G70</td>
</tr>
<tr>
<td>3.0 GHz Core2 Duo</td>
<td>3.0 GHz Core2 Duo</td>
</tr>
</tbody>
</table>

GT200 = GeForce GTX 280
G92 = GeForce 9800 GTX
G80 = GeForce 8800 GTX
G71 = GeForce 7900 GTX
NV35 = GeForce FX 5950 Ultra
NV30 = GeForce FX 5800
NV40 = GeForce 6800 Ultra
G70 = GeForce 7800 GTX
Message:

“High-level programming (through abstraction) eases the programmability problem posed by new architectures without sacrificing high performance”

- FLAME
- PLAPACK
OUTLINE

- Evaluation and tuning of CUBLAS
- Superscalar techniques in the construction of dense linear algebra libraries for multi-GPU platforms:
  1. Data-flow dynamic scheduling
  2. DSM
- Clusters of GPUs
OUTLINE

- Evaluation and tuning of CUBLAS
- Superscalar techniques in the construction of dense linear algebra libraries for multi-GPUs platforms:
  1. Data-flow dynamic scheduling
  2. DSM
- Clusters of GPUs
CUBLAS

NVIDIA cublasSgemm (2.3) on a Tesla C1060 (240 cores)

38% peak
CUBLAS

NVIDIA Level-3 CUBLAS (2.3) on a Tesla C1060 (240 cores)

- cublasSgemm
- cublasSsyr2k
- cublasSsymm

GFLOPS

Problem size (m=n=k)

38% peak
25% peak
22% peak
TUNED CUBLAS USING FLAME

Cast Symm in terms of Gemm and minor Symm

\[
C = A \times B
\]

\[
\begin{array}{c}
C_{11} \\
A_{10} \quad A_{11} \\
\end{array}
\]

\[
\begin{array}{c}
B_{01} \\
B_{11} \\
\end{array}
\]
TUNED CUBLAS USING FLAME

NVIDIA Level-3 CUBLAS (2.3) on a Tesla C1060 (240 cores)

GFLOPS

Problem size (m=n=k)

3X
while ( FLA_Obj_length( ATL ) < FLA_Obj_length( A ) ) {
    /* 2x2->3x3 repartitionings of A, B and C */
    FLA_Gemm( ..., A10, B01, ..., C11 );
    FLA_Symm( ..., A11, B11, ..., C11 );
    /* 3x3->2x2 repartitionings of A, B and C */
}
OUTLINE

- Evaluation and tuning of CUBLAS
- Superscalar techniques in the construction of dense linear algebra libraries for multi-GPU platforms:
  1. Data-flow dynamic scheduling
  2. DSM
- Clusters of GPUs
MULTI-GPU PLATFORMS

- CPU-hardware accelerators
  - More favourable price-performance ratio
- Slow communication between host and devices
- Separate memory spaces: no hardware coherence
DATA-FLOW DYNAMIC SCHEDULING

- Out-of-order execution controlled by data dependencies (*data-flow parallelism at task level*)

- Goals:
  - Increase the degree of parallelism during the execution of dense linear algebra operations
  - Balance the workload distribution
  - Hide dynamic scheduling in an architecture-dependent run-time: ease programmability
DATA-FLOW DYNAMIC SCHEDULING

Current libraries

- Cholesky factorization

\[ A = L \times L^T \]

Key to the solution of (s.p.d.) linear systems

\[ A \ x = b \equiv (L \ L^T) \ x = b \]

\[ L \ y = b \ \Rightarrow \ y \]

\[ L^T \ x = y \ \Rightarrow \ x \]
DATA-FLOW DYNAMIC SCHEDULING

Current libraries

- Blocked algorithm cast in terms of Gemm

\[ A_{11} = L_{11} \times L_{11}^T \]

\[ L_{21} \leftarrow A_{21} \times L_{11}^{-T} \]

\[ A_{22} \leftarrow A_{22} - L_{21} \times L_{21}^T \]

Multi-core processor: multithreaded implementation of T and P
DATA-FLOW DYNAMIC SCHEDULING
Current libraries

- Blocked algorithm cast in terms of Gemm
DATA-FLOW DYNAMIC SCHEDULING

Current libraries

Cholesky factorization on 2 Xeon QuadCore (8 cores)

Problem size

GFLOPS

8 Xeon cores
4 Xeon cores
1 Xeon core

57% peak
71% peak
80% peak
DATA-FLOW DYNAMIC SCHEDULING
Current libraries

- Why?
There is more parallelism than is being exploited

In the same iteration
In different iterations

1st iteration
2nd iteration
DATA-FLOW DYNAMIC SCHEDULING

- Read/written blocks determine dependencies

```c
while ( ... ) {
    /* 2x2->3x3 repartitioning */
    FLA_Chol( A11 );
    FLA_Trsm( ..., A11, A21 );
    FLA_Syrk( ..., A21, ..., A22 );
    /* 3x3->2x2 repartitioning */
}
```

Task tree as a DAG:
DATA-FLOW DYNAMIC SCHEDULING

- Execution of task tree:
  - Scheduling (temporal) dictated by data dependencies
  - Cache-aware mapping (spatial)
Outline

- Evaluation and tuning of CUBLAS
- Superscalar techniques in the construction of dense linear algebra libraries for multi-core processors and GPUs:
  1. Data-flow dynamic scheduling
  2. DSM
- Clusters of GPUs
DISTRIBUTED-SHARED MEMORY

- Middleware that deals with separate address spaces in host and each device (GPU)

- Goals:
  - Reduce the number of data transfers: increase efficiency
  - Hide the existence of multiple memory spaces: ease programmability
DISTRIBUTED-SHARED MEMORY

- Data transfer
  - Before execution, transfer data to device
  - Upon completion, retrieve results back to host

→ poor data locality
DISTRIBUTED-SHARED MEMORY

- Shared memory system

Multi-core processor with hw. coherence:

\[ \text{MP} \rightarrow \text{P}_0 + C_0 \]
\[ \text{MP} \rightarrow \text{P}_1 + C_1 \]
\[ \text{MP} \rightarrow \text{P}_2 + C_2 \]
\[ \text{MP} \rightarrow \text{P}_3 + C_3 \]

Multi-GPU platform
DISTRIBUTED-SHARED MEMORY

- Reduce #data transfers
  - Software cache in devices:
    - Operate at block level
    - Software $\rightarrow$ flexibility
    - Write-back
    - Write-invalidate

Multi-GPU platform
D.-F. DYNAMIC SCHEDULING + DSM
Performance in multi-GPU platforms

![Graph showing performance comparison between different libraries on Tesla S1070 (4 GT200) compared to other platforms.](image)

- **libflame on 4 GT200 (4x240 cores)**
- **cublasSgemm on 1 GT200 (240 cores)**
- **MKL sgemm on 1 Intel Xeon QuadCore (4 cores)**

**GFLOPS vs Matrix size (m=n=k)**

- **1.2 TFLOPS**
- **3.15X**
D.-F. DYNAMIC SCHEDULING + DSM
Performance in multi-GPU platforms

Cholesky factorization on Tesla S1070 (4 GT200)

libflame on 4 GT200 (4x240 cores)
MKL spotrf on 2 Intel Xeon QuadCore (8 cores)
MULTI-GPU PLATFORMS

- Scheduling and separate memory address spaces hidden under the covers (run-time)
- No changes to libflame!
OUTLINE

- Evaluation and tuning of CUBLAS
- Superscalar techniques in the construction of dense linear algebra libraries for multi-core processors and GPUs:
  1. Data-flow dynamic scheduling
  2. DSM
- Clusters of GPUs
CLUSTERS of GPUs

- CPU-hardware accelerators
  - More favourable price-performance ratio
  - Multi-GPU have limited scalability
CLUSTERS of GPUs

- PLAPACK
  - Message-passing dense linear algebra
  - Object-based approach, like libflame
  - Communication cleanly separated from computation
  - Copies between objects with different distributions: PLA_Copy & PLA_Reduce
CLUSTERS of GPUs

- Reduce #data transfers
  - Keep data in device memory:
    - Transfer data to main memory only during communication
    - Use communication packing to transfer
CLUSTERS of GPUs

Performance

Sgemm on cluster of FX5800

- PLAPACK on 32 FX5800
- PLAPACK on 16 FX5800
- PLAPACK on 8 FX5800
- cublasSgemm

8 TFLOPS
22X
5.5X
10X
CLUSTERS of GPUs
Performance

CUPLAPACK. Cholesky factorization on longhorn

<table>
<thead>
<tr>
<th>Configuration</th>
<th>GFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLAPACK on 32 FX5800</td>
<td></td>
</tr>
<tr>
<td>PLAPACK on 16 FX5800</td>
<td></td>
</tr>
<tr>
<td>PLAPACK on 8 FX5800</td>
<td></td>
</tr>
<tr>
<td>PLAPACK on 1 FX5800</td>
<td></td>
</tr>
</tbody>
</table>

Matrix size

0 10000 20000 30000 40000 50000 60000 70000 80000 90000 100000

GFLOPS

0 1000 2000 3000 4000 5000 6000 7000 8000 9000 10000
CLUSTERS of GPUs

while ( TRUE ) {
    /* Split ABR into 2x2 views */
    PLA_Local_Chol( A11 );
    PLA_Trsm( ..., A11, A21 );
    PLA_Syrk( ..., A21, ..., A22 );
}

- Separate memory address spaces hide under the communication routines
- No other changes to PLAPACK!
“High-level programming (through abstraction) eases the programmability problem posed by new architectures without sacrificing high performance”

- FLAME
- PLAPACK
CONCLUSIONS

Dilated experience with linear algebra on GPUs:


▪ "Retargeting PLAPACK to Clusters with Hardware Accelerators." FLAME WN #42, UTCS TR-10-06, Feb. 2010.

Thanks for your attention!

▪ For more information: www.cs.utexas.edu/users/flame