Shooting for the Stars with GPUs

Hatem Ltaief
Extreme Computing Research Center

KAUST-NVIDIA Workshop

February 17, 2014
King Abdullah University of Science and Technology
Thuwal, Saudi Arabia
Outline

1 Motivations

2 Task-based Programming Model

3 Application: The European Extremely Large Telescope

4 Parallel Implementation

5 Performance Results

6 Conclusion
Students/Collaborators/Support

- Extreme Computing Research Center @ KAUST: A. Abdelfattah, **A. Charara** and D. Keyes
- Observatoire de Paris, LESIA: E. Gendrony, D. Gratadour, C. Morely, A. Sevin and F. Vidal
- KAUST IT Hardware support
Outline

1. Motivations
2. Task-based Programming Model
3. Application: The European Extremely Large Telescope
4. Parallel Implementation
5. Performance Results
6. Conclusion
Motivations

Today’s Top500 Fastest System: Tianhe-2

- 34 Pflop/s, more than 170 times Shaheen!
- High level of concurrency: more than 3,000,000 cores (Intel Xeon x86 + Phi)
- Bulk synchronous parallel model, RIP!
- 18 MW needed to feed the baby
- Exascale roadmap says up to 20 MW Power Envelope
- Huge challenge: achieving 2 orders of magnitude in performance with roughly the same power envelope
- Co-designed Hardware and Software solutions
Motivations

A Look Back from DLA Libraries’ Perspective...

Software infrastructure and algorithmic design follow hardware evolution in time:

- **70’s - LINPACK**, vector operations:  
  *Level-1 BLAS operation*
A Look Back from DLA Libraries’ Perspective...

Software infrastructure and algorithmic design follow hardware evolution in time:

- **70’s - LINPACK, vector operations:**
  - *Level-1 BLAS operation*

- **80’s - LAPACK, block, cache-friendly:**
  - *Level-3 BLAS operation*
Motivations

A Look Back from DLA Libraries’ Perspective...

Software infrastructure and algorithmic design follow hardware evolution in time:

- 70’s - LINPACK, vector operations:
  *Level-1 BLAS operation*

- 80’s - LAPACK, block, cache-friendly:
  *Level-3 BLAS operation*

- 90’s - ScaLAPACK, distributed memory:
  *PBLAS Message passing*
A Look Back from DLA Libraries’ Perspective...

Software infrastructure and algorithmic design follow hardware evolution in time:

- 70’s - LINPACK, vector operations: 
  
  *Level-1 BLAS operation*

- 80’s - LAPACK, block, cache-friendly: 
  
  *Level-3 BLAS operation*

- 90’s - ScaLAPACK, distributed memory: 
  
  *PBLAS Message passing*

- 00’s:
  
  - PLASMA, MAGMA, many x86/cuda cores friendly: 
    
    *DAG scheduler, tile data layout, some extra kernels*
Outline

1 Motivations

2 Task-based Programming Model

3 Application: The European Extremely Large Telescope

4 Parallel Implementation

5 Performance Results

6 Conclusion
One of The Possible Solution for Exascale Computing

- Fine-granularity
- Local data access
- Local synchronization
- Runtime system for separation of concerns
- Productivity with abstraction
- Performance/debugging tools
- Popular: available from OpenMP 3.0
General Procedure

1. "Taskify" the application. This may require:
   - Implementing a new algorithm
   - Performing more flops at the end
   - Increasing code size

2. Schedule the generated tasks. This may require a runtime system featuring:
   - Static/dynamic scheduling
   - Shared/Distributed memory systems
   - NUMA-aware
   - Heterogeneous architecture (x86+accelerators)
Many Success Stories

- Dense Linear Algebra (PLASMA/MAGMA/PaRSEC, FLAME).
- Fast Multipole Method (Agullo et al. in SIAM SISC 2014, Ltaief and Yokota in Conc. and Comp. 2013, etc...).
- Sparse iterative solvers (Ghysels et al. SIAM SISC 2013).
- Parareal (Elwasif et al. JCP 2012).

At Petascale, the task-based Uintah framework (Meng et al. SC’13) handled various sparse applications:

- Fluid-structure interaction with adaptive mesh refinement.
- Radiation modeling through raytracing.
- Turbulent combustion on a fixed-mesh requiring large-scale linear solves.
Outline

1. Motivations
2. Task-based Programming Model
3. Application: The European Extremely Large Telescope
4. Parallel Implementation
5. Performance Results
6. Conclusion
The World’s Biggest Eye on The Sky

Credits: ESO (http://www.eso.org/public/teles-instr/e-elt/)
- The largest optical/near-infrared telescope in the world.
- A highest priority in ground-based astronomy.
- It weighs about 2700 tons.
- The main mirror diameter is 39m.
- Location in Chile, South America.
- E-ELT does not exist yet, expected early 2020s.
Designing the E-ELT Instruments

Simulation!
Simulation!
Simulation!
Hmmm... Did I say Simulation?
Multi-object Adaptive Optics (MOAO)

- Probably the most challenging embedded instrument in the E-ELT.
- Observe/understand the evolution of a number of the most distant galaxies in parallel.
- Capable of exploiting the Field of View (FoV) of 7 to 10 arcminutes.
- It is used on telescopes to compensate, in **real-time**, for the effect of atmospheric turbulence, providing a significant improvement in resolution and in return, to adjust the deformable mirrors that compensate for these distortions before sending the light to the science instrument.
- Good news: extremely compute intensive at full scale.
Global Workflow Chart

- System parameters
  - matcov
  - cmm
  - cpm
  - cpp

- Turbulence parameters
  - matcov
  - cmm
  - cpm
  - cpp

- EVD
  - mca

- ToR
  - R

- BLAS
  - cee
  - cvv
  - Inter-sample
  - PSF
Global Workflow Chart
The Tomographic Reconstructor ToR
The Tomographic Reconstructor ToR

- Covariance matrix generation
- Symmetric Matrix inversion
- Symmetric matrix-matrix multiplication
The Tomographic Reconstructor ToR

- Covariance matrix generation
- **Symmetric Matrix inversion**
- **Symmetric matrix-matrix multiplication**
Outline

1. Motivations
2. Task-based Programming Model
3. Application: The European Extremely Large Telescope
4. Parallel Implementation
5. Performance Results
6. Conclusion
Two Algorithmic Variants for Matrix Inversion

- Pseudo-inverse using spectral decomposition (with blocked algorithm i.e., OpenMP)
- Cholesky-based matrix inversion (with tile algorithm i.e., Task-based)
**Blocked Algorithms**

![Diagram](image)

(a) First step.  
(b) Second step.  
(c) Third step.

**Figure**: Panel-update sequences for the LAPACK two-sided transformations.
Blocked Algorithms: Fork-Join Paradigm
Blocked Algorithms

Principles:

- Panel-Update Sequence
- Transformations are blocked/accumulated within the Panel (Level 2 BLAS)
- Transformations applied at once on the trailing submatrix (Level 3 BLAS)
- Parallelism hidden inside the BLAS
- Fork-join Model
Tile Data Layout Format

LAPACK: column-major format

PLASMA: tile format
PLASMA: Parallel Linear Algebra for Scalable Multi-core Architectures
⇒ http://icl.cs.utk.edu/plasma/

- Parallelism is brought to the fore
- May require the redesign of linear algebra algorithms
- Tile data layout translation
- Remove unnecessary synchronization points between Panel-Update sequences
- DAG execution where nodes represent tasks and edges define dependencies between them
- Default dynamic runtime system environment QUARK (but could use StarPU, DAGUE/PaRSEC, SuperMatrix, ParalleX, OMPSs etc.)
StarPU Runtime System

- RunTime which provides:
  - Task scheduling
  - Memory management

- Supports:
  - SMP/Multicore Processors (x86, PPC, ...)
  - NVIDIA GPUs (e.g. heterogeneous multi-GPU)
  - OpenCL devices
  - Cell Processors (experimental)
Parallel Implementation

StarPU Runtime System

```c
starpu_Insert_Task(&cl_dpotrf,
    VALUE, &uplo, sizeof(char),
    VALUE, &n, sizeof(int),
    INOUT, Ahandle(k, k),
    VALUE, &lda, sizeof(int),
    OUTPUT, &info, sizeof(int),
    CALLBACK, profiling?cl_dpotrf_callback:NULL, NULL, 0);
```
Outline

1. Motivations
2. Task-based Programming Model
3. Application: The European Extremely Large Telescope
4. Parallel Implementation
5. Performance Results
6. Conclusion
Performance Results

ToR with Pseudo-Inverse Using Eigenvalue Decomposition (Blocked Algorithm)

Time breakdown in seconds of the tomographic reconstructor for \(N=40,000\): **60% speedup**

<table>
<thead>
<tr>
<th></th>
<th>16 cores</th>
<th>8 GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intel SB</td>
<td>Kepler K20c</td>
</tr>
<tr>
<td>DSYEVD</td>
<td>4370.50</td>
<td>399.15</td>
</tr>
<tr>
<td>Pseudo inverse</td>
<td>236.70</td>
<td>18.66</td>
</tr>
<tr>
<td>DGEMM</td>
<td>49.05</td>
<td>3.29</td>
</tr>
<tr>
<td>Total</td>
<td>4656.25</td>
<td>421.10</td>
</tr>
</tbody>
</table>

ToR Cholesky-based Matrix Inversion (Tile Algorithm)

Performance of the TR simulation in time (seconds)
ToR Cholesky-based Matrix Inversion (Tile Algorithm)

Performance comparisons against previous ToR implementation (blocked algorithm)

Performance Results

ToR Cholesky-based Matrix Inversion (Tile Algorithm)

Performance of the TR simulation in GFlop/s

- 4 GPUs + 16 Cores
- 3 GPUs + 16 Cores
- 2 GPUs + 16 Cores
- 1 GPU + 16 Cores
- 16 Cores
ToR Cholesky-based Matrix Inversion (Tile Algorithm)

TR Performance assessment

![Graph showing TR performance assessment with different matrix sizes and performance metrics. The graph compares theoretical peak performance and actual performance for different matrix sizes. The legend includes lines for Theoretical-PEAK, GEMM-PEAK, 4 GPUs MORSE-GEMM, and 4 GPUs TOR.](attachment:image.png)
ToR Cholesky-based Matrix Inversion (Tile Algorithm)

TR simulation trace showing decent utilization of 16 cores with 4 GPUs
Directed Acyclic Graph for each ToR’s stage
Outline

1. Motivations
2. Task-based Programming Model
3. Application: The European Extremely Large Telescope
4. Parallel Implementation
5. Performance Results
6. Conclusion
Summary

- Breakthrough for the computational astronomy (*Nature Middle East Highlight*, **KAUST Discovery**).
- Simulation close to real-time.
- Capable of simulating hours of observations in few seconds.
- Efficient Task-based programming model
- Pipelining computational stages
- Leverage to other architectures e.g., ARM processors (energy savings)
- Leverage to distributed memory systems
Thank You!

شكرا!