Recipe for High Performance Computing
Software and Application at Exascale

Hatem Ltaief
KAUST Supercomputing Lab

HPC Advisory Council Switzerland Conference

March 13-15, 2012
Lugano, Switzerland
Credits

HPC Advisory Council
Network of Expertise
Outline

1. Motivations
2. Matrices Over Runtime Systems at Exascale
3. Cholesky-based Matrix Inversion and Generalized Symmetric Eigenvalue Problem
4. N-Body Simulations
5. Seismic Applications
6. Conclusion
Outline

1. Motivations
2. Matrices Over Runtime Systems at Exascale
3. Cholesky-based Matrix Inversion and Generalized Symmetric Eigenvalue Problem
4. N-Body Simulations
5. Seismic Applications
6. Conclusion
The "K" Computer

**K computer Specifications**

<table>
<thead>
<tr>
<th>CPU (SPARC64 VIII/fx)</th>
<th>Cores/Node</th>
<th>8 cores (@2GHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>128GFlops</td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td>SPARC V9 + HPC extension</td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>L1(I/D) Cache : 32KB/32KB, L2 Cache : 6MB</td>
<td></td>
</tr>
<tr>
<td>Power</td>
<td>58W (typ. 30 C)</td>
<td></td>
</tr>
<tr>
<td>Mem. bandwidth</td>
<td>64GB/s.</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>1 CPU / Node</td>
<td></td>
</tr>
<tr>
<td>Memory capacity</td>
<td>16GB (2GB/core)</td>
<td></td>
</tr>
<tr>
<td>System board(SB)</td>
<td>No. of nodes</td>
<td>4 nodes /SB</td>
</tr>
<tr>
<td>Rack</td>
<td>No. of SB</td>
<td>24 SBs/rack</td>
</tr>
<tr>
<td>System</td>
<td>Nodes/system</td>
<td>&gt; 80,000</td>
</tr>
</tbody>
</table>

**Inter-connect**

<table>
<thead>
<tr>
<th>Topology</th>
<th>6D Mesh/Torus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>5GB/s. for each link</td>
</tr>
<tr>
<td>No. of link</td>
<td>10 links/ node</td>
</tr>
<tr>
<td>Additional feature</td>
<td>H/W barrier, reduction</td>
</tr>
<tr>
<td>Architecture</td>
<td>Routing chip structure (no outside switch box)</td>
</tr>
<tr>
<td>Cooling</td>
<td>CPU, ICC*</td>
</tr>
<tr>
<td>Other parts</td>
<td>Direct water cooling</td>
</tr>
<tr>
<td></td>
<td>Air cooling</td>
</tr>
</tbody>
</table>

**System**

- LiNPACK 10 PFlops over 1PB mem.
- 800 racks
- 80,000 CPUs
- 640,000 cores

**Node**

- 128 GFlops
- 16GB Memory
- 64GB/s Memory bandwidth

**System Board**

- 512 GFlops
- 64 GB memory

**Rack**

- 12.3 TFlops
- 15TB memory

* ICC : Interconnect Chip
HPC Recipe For Exascale Computing

Dynamic runtime systems

Fine-grain parallelism

Auto-Tuning

Data Motion Reducing

Power efficiency

Synchronization Reducing
HPC Recipe For Exascale Computing

Dynamic runtime systems

Fine-grain parallelism

Power efficiency

Auto-Tuning

Data Motion Reducing

Synchronization Reducing

Hardware and Software Co-design

H. Ltaief
HPC Recipe For Exascale Computing

- Dynamic runtime systems
- Auto-Tuning
- Data Motion Reducing
- Synchronization Reducing
- Power efficiency
- Fine-grain parallelism

Hardware and Software Co-design

10^{18}
A Look Back...

Software infrastructure and algorithmic design follow hardware evolution in time:

- 70’s - LINPACK, vector operations:
  
  *Level-1 BLAS operation*
Software infrastructure and algorithmic design follow hardware evolution in time:

- **70’s - LINPACK**, vector operations:
  \> \textit{Level-1 BLAS operation}

- **80’s - LAPACK**, block, cache-friendly:
  \> \textit{Level-3 BLAS operation}
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...

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*
Block Algorithms

- 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
One-Sided Block Algorithms: LU
Block Algorithms: Fork-Join Paradigm
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
- Remove unnecessary synchronization points between Panel-Update sequences
- DAG execution where nodes represent tasks and edges define dependencies between them
- Dynamic runtime system environment QUARK
- Tile data layout translation
Tile Data Layout Format

LAPACK: column-major format

PLASMA: tile format
MAGMA: Matrix Algebra on GPU and Multicore Architectures

http://icl.cs.utk.edu/magma

- Lessons Learned from PLASMA!
- CUDA-based hybrid systems
- New high performance numerical kernels
- StarPU Runtime System (INRIA, Bordeaux)
- Both: x86 and GPUs → Hybrid Computations
- Similar to LAPACK in functionality
Dynamic Runtime System

• Conceptually similar to out-of-order processor scheduling because it has:
  • Dynamic runtime DAG scheduler
  • Out-of-order execution flow of fine-grained tasks
  • Task scheduling as soon as dependencies are satisfied
  • Producer-Consumer
DataFlow Programming

- Five decades **OLD** concept
- Programming paradigm that models a program as a directed graph of the data flowing between operations (cf. Wikipedia)
- Think "how things connect" rather than "how things happen"
- *Assembly line*
- Inherently parallel
Outline

1 Motivations

2 Matrices Over Runtime Systems at Exascale

3 Cholesky-based Matrix Inversion and Generalized Symmetric Eigenvalue Problem

4 N-Body Simulations

5 Seismic Applications

6 Conclusion
Matrices Over Runtime Systems at Exascale

- Mission statement: "Design dense and sparse linear algebra methods that achieve the fastest possible time to an accurate solution on large-scale Hybrid systems".
- Runtime challenges due to the ever growing hardware complexity.
- Algorithmic challenges to exploit the hardware capabilities at most.
QUARK

- From Sequential Nested-Loop Code to Parallel Execution
- Task-based parallelism
- Out-of-order dynamic scheduling
- Scheduling a Window of Tasks
- Data Locality and Cache Reuse
- High user-productivity
- Shipped within PLASMA but standalone project
QUARK

User Code
- Insert Task T1
- Insert Task T2
- Insert Task T3
- Insert Task T4
- Insert Task T5
- Insert Task T6
- Insert Task T7
- Insert Task T8

Master Thread
- Inserting tasks;
- Determining dependencies;
- Queuing tasks

Worker Threads
- Finding tasks;
- Executing task;
- Checking descendants

Worker Queue: T3
Worker Queue: T5
Worker Queue:
int QUARK_core_dpotrf( Quark *quark, char uplo, int n, double *A, int lda, int *info )
{
    QUARK_Insert_Task( quark, TASK_core_dpotrf, 0x00,
    sizeof(char), &uplo, VALUE,
    sizeof(int), &n, VALUE,
    sizeof(double)*n*n, A, INOUT | LOCALITY,
    sizeof(int), &lda, VALUE,
    sizeof(int), info, OUTPUT,
    0);
}

void TASK_core_dpotrf(Quark *quark)
{
    char uplo; int n; double *A; int lda; int *info;
    quark_unpack_args_5( quark, uplo, n, A, lda, info );
    dpotrf_( &uplo, &n, A, &lda, info );
}
StarPU

- **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)
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);
SMPSs

- Compiler technology.
- Task parameters and directionality defined by the user through pragmas
- Translates C codes with pragma annotations to standard C99 code
- Embedded Locality optimizations
- Data renaming feature to reduce dependencies, leaving only the true dependencies.
#pragma css task input(A[NB][NB]) inout(T[NB][NB])
void dsyrk(double *A, double *T);

#pragma css task inout(T[NB][NB])
void dpotrf(double *T);

#pragma css task input(A[NB][NB], B[NB][NB]) inout(C[NB][NB])
void dgemm(double *A, double *B, double *C);

#pragma css task input(T[NB][NB]) inout(B[NB][NB])
void dtrsm(double *T, double *C);

#pragma css start
for (k = 0; k < TILES; k++) {
    for (n = 0; n < k; n++)
        dsyrk(A[k][n], A[k][k]);
    dpotrf(A[k][k]);

    for (m = k+1; m < TILES; m++) {
        for (n = 0; n < k; n++)
            dgemm(A[k][n], A[m][n], A[m][k]);
        dtrsm(A[k][k], A[m][k]);
    }
}
#pragma css finish
Standardization???

- Efforts to define an API standard for these runtime systems.
- Difficult task...
- But worth the time and sacrifice when it comes to making end users life easier.
DAGuE: Directed Acyclic Graph Unified Environment
⇒ http://icl.cs.utk.edu/dague
- Compiler technology.
- Converting Sequential Code to a DAG representation.
- Parametrized DAG scheduler for distributed memory systems.
- Engine of DPLASMA library
Outline

1. Motivations
2. Matrices Over Runtime Systems at Exascale
3. Cholesky-based Matrix Inversion and Generalized Symmetric Eigenvalue Problem
4. N-Body Simulations
5. Seismic Applications
6. Conclusion
A^{-1}, Seriously???

- YES!
- A is a dense symmetric matrix
- Three steps:
  1. Cholesky factorization (DPOTRF)
  2. Inverting the Cholesky factor (DTRTRI)
  3. Calculating the product of the inverted Cholesky factor with its transpose (DLAUUM)
- StarPU runtime used here
A⁻¹, Hybrid Architecture Targeted

- PCI Interconnect 16X 64Gb/s, very thin pipe!
- Fermi C2050 448 cuda cores 515 Gflop/s
How to choose NB (Tile size)? **Auto-Tuning???:)**
Auto-Tuning

CUDA GPU Roadmap

- Maxwell
- Kepler
- Fermi
- Tesla

DP GFLOPS per Watt

- 16
- 14
- 12
- 10
- 8
- 6
- 4
- 2

- 2007
- 2009
- 2011
- 2013
Auto-Tuning

CUDA GPU Roadmap

*Slide courtesy of J. Kurzak, UTK*
$A^{-1}$, Preliminary Results

H. Ibeid, D. Kaushik, D. Keyes and H. Ltaief, HIPC’11, India
GSEVP: What we solve?

\[ Ax = \lambda Bx \]

- \( A, B \in \mathbb{R}^{n \times n}, \ x \in \mathbb{R}^n, \ \lambda \in \mathbb{R} \) or \( A, B \in \mathbb{C}^{n \times n}, \ x \in \mathbb{C}^n, \ \lambda \in \mathbb{R} \)

- \( A = A^T \) or \( A = A^H \)
- \( B \) is symmetric positive definite
GSEVP: Why we solve it?

To obtain energy eigenstates in:

- Chemical cluster theory
- Electronic structure of semiconductors
- Ab-initio energy calculations of solids
GSEVP: How to solve it?

\[ Ax = \lambda Bx \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Explanation</th>
<th>LAPACK routine name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( B = L \times L^T )</td>
<td>Cholesky factorization</td>
<td>POTRF</td>
</tr>
<tr>
<td>2 ( C = L^{-1} \times A \times L^{-T} )</td>
<td>application of triangular factors or HEGST</td>
<td>SYGST</td>
</tr>
<tr>
<td>3 ( T = Q^T \times C \times Q )</td>
<td>tridiagonal reduction</td>
<td>SYEVD or HEEVD</td>
</tr>
<tr>
<td>4 ( Tx = \lambda x )</td>
<td>QR iteration</td>
<td>STERF</td>
</tr>
</tbody>
</table>
All computational stages: separately
All computational stages: combined

- Dependencies are tracked inside PLASMA by QUARK.
Combining stages: matrix view
Results on 4-socket AMD Magny Cours (48 cores)

H. Ltaief, P. Luszczek, A. Haidar, J. Dongarra – ParCo’11, Belgium
Power Monitoring with PowerPack
Power Rate of LAPACK TRD

H. Ltaief, P. Luszczek, J. Dongarra – EnaHPC’11, Germany
Power Rate of PLASMA TRD

H. Ltaief, P. Luszczek, J. Dongarra – EnaHPC’11, Germany
QUARK and DVFS
Outline

1. Motivations
2. Matrices Over Runtime Systems at Exascale
3. Cholesky-based Matrix Inversion and Generalized Symmetric Eigenvalue Problem
4. N-Body Simulations
5. Seismic Applications
6. Conclusion
Astrophysics and Turbulence Applications w/ R. Yokota

Pseudo Spectral Method
(2/3 dealiasing)
Re_κ: 500
N : 4096^3

Vortex Particle Method
(Reinitialized CSM)
Re_κ: 500
N : 4096^3

64 billion particles
Data Driven Fast Multipole Method

- M2M
- P2M
- M2L
- L2L
- L2P
- P2P

blue data depends on red

source particles

target particles
Dual Tree Traversal
Adjustable Granularity

breadth first

coarse grained tasks

no mutual interaction
Strong Scaling w/ QUARK

H. Ltaief and R. Yokota – submitted to EuroPar’12
Strong Scaling w/ QUARK

H. Ltaief and R. Yokota – submitted to EuroPar’12
What is next?

- Heterogeneous architecture w/ hardware accelerators (StarPU)
- Distributed memory systems (DAGuE)
- Implementation of the reduction operation
Outline

1. Motivations
2. Matrices Over Runtime Systems at Exascale
3. Cholesky-based Matrix Inversion and Generalized Symmetric Eigenvalue Problem
4. N-Body Simulations
5. Seismic Applications
6. Conclusion
Seismic Acquisition
Standard Stencil Time Integration

- Reverse Time Migration
- Acoustic Wave Equation with constant density
- Numerical solution: Finite Difference
- Isotropic Case: Axis directions (X,Y,Z)
- Tilted Transverse Isotropic (TTI) Case: Axis directions (X,Y,Z) & Cross Derivatives (XY, YZ, XZ)
Standard Stencil Time Integration

- Isotropic stencil

- TTI stencil

*Slide courtesy of S. Feki, TOTAL Houston*
Standard Stencil Time Integration

- Explicit time integration scheme with domain decomposition
- Order one in space and time
Standard Stencil Time Integration

- Explicit time integration scheme with domain decomposition
- Order one in space and time
Standard Stencil Time Integration

- Explicit time integration scheme with domain decomposition
- Order one in space and time
Standard Stencil Time Integration

- Explicit time integration scheme with domain decomposition
- Order one in space and time

Diagram showing time steps and halo regions for GPUs GPU0 and GPU1.
Standard Stencil Time Integration

- Explicit time integration scheme with domain decomposition
- Order one in space and time
Standard Stencil Time Integration

- Explicit time integration scheme with domain decomposition
- Order one in space and time
New Stencil Time Integration

- Communication-avoiding by reducing halo exchanges frequency
- Synchronization-reducing by instruction reordering
New Stencil Time Integration

- Communication-avoiding by reducing halo exchanges frequency
- Synchronization-reducing by instruction reordering
New Stencil Time Integration

- Communication-avoiding by reducing halo exchanges frequency
- Synchronization-reducing by instruction reordering
New Stencil Time Integration

- Communication-avoiding by reducing halo exchanges frequency
- Synchronization-reducing by instruction reordering
New Stencil Time Integration

- Communication-avoiding by reducing halo exchanges frequency
- Synchronization-reducing by instruction reordering
New Stencil Time Integration

- Communication-avoiding by reducing halo exchanges frequency
- Synchronization-reducing by instruction reordering
New Stencil Time Integration

- Communication-avoiding by reducing halo exchanges frequency
- Synchronization-reducing by instruction reordering
Stencil kernels

- Seismic kernels are typically order 2 in time and order 8 in space
- Two phases: calculate solutions inside a cone and then outside the cone
- The cone kernel becomes the fine-grain task to dynamically schedule
- No numerical instabilities added to the original scheme
- May have load imbalance due to PML high compute intensity
- Similar (to some extend) to Demmel’s approach on the Matrix power kernel.
- Work in progress with A. Abdelfattah, PhD Student and Saudi Aramco
Outline

1. Motivations
2. Matrices Over Runtime Systems at Exascale
3. Cholesky-based Matrix Inversion and Generalized Symmetric Eigenvalue Problem
4. N-Body Simulations
5. Seismic Applications
6. Conclusion
• Dataflow programming models could play a major role for exascale challenges
• Need efficient runtime with high productivity *in mind*
• Need new flexible/asynchronous algorithms
• Work potentially for dense as well as sparse computations
• Need to determine an appropriate granularity to hide scheduling overhead
Just in case you are wondering what's beyond ExaFlops...

Mega, Giga, Tera, Peta, Exa, Zetta ...

- $10^3$ kilo
- $10^6$ mega
- $10^9$ giga
- $10^{12}$ tera
- $10^{15}$ peta
- $10^{18}$ exa
- $10^{21}$ zetta

- $10^{24}$ yotta
- $10^{27}$ xona
- $10^{30}$ weka
- $10^{33}$ vunda
- $10^{36}$ uda
- $10^{39}$ treda
- $10^{42}$ sorta
- $10^{45}$ rinta
- $10^{48}$ queixa
- $10^{51}$ pepta
- $10^{54}$ ocha
- $10^{57}$ nena
- $10^{60}$ minga
- $10^{63}$ luma
Thank you!

شكراً!