Lessons From Tuning GEMM on CPU and GPU

Aug. 12, 2009
Charles Fu
Why Is It So Hard?
Why Is It So Hard (cont)?

• On CPU

Data
- aligned/misaligned
- leading dimension odd/even
- small size/large size

Cache
- size
- hierarchy (L1, L2, L3)
- replacement policy
- associativity
- bandwidth

TLB
- size
- associativity

Instruction
- latency
- throughput
- issue rate
- issue port
- branch

Spec
- copy C or not
- loop order (N, K, M, ...)
- loop blocking
- (which, depth)
- prefetch
- (where, size, dist, type)
- packing
- (A, B, or C)
- insn selection
- insn scheduling
- loop unrolling
- software pipeline
Why Is It So Hard (cont)?

- On GPU

**Data**
- small size / large size

**Machine Model**
- Cache
  - register size
  - share mem size
  - device mem size
  - bandwidth
- Runtime
  - warp size
  - threadblock size
- Instruction
  - issue rate
  - branch

**Spec**
- in-core/out-of-core padding
- loop order (M,N)
- streaming copy
- loop order (N,K,M)
- loop blocking
- packing (where, size, explicitly)
- transpose in place
- insn selection
- insn scheduling
- loop unrolling
- software pipeline

Copyright Microsoft Corporation, All rights reserved
Why Is It So Hard (cont)?

• On GPU:

Object Function:
Fastest execution

Constraints:
MB x NB x sizeof(type) x max_threadblks ≤ register_file -- reg_constraint
MB x KB x sizeof(type) x max_threadblks ≤ share_mem -- shmem_constraint
NB = num_threads/warp x p -- sched_constraint
(MB x KB + KB x NB + MB x NB) x sizeof(type) x max_threadblks ≤ device_mem

while p is integer

GTX280:
MB x NB ≤ 4k
MB x KB ≤ 1k
NB = {32, 64, 128}
Tuning Parallel GEMM

- Sequential Kernel is the key for the performance
- Blocking parameters to be tuned
Lesson 1: 80/20 Rule

• To get close to peak performance, every piece of tuning has to be perfect.
• 80% time and effort spent on hand-tuned kernel
• To write and run micro-benchmarks to probe unclear microarchitecture issues on CPU
• To change data layout for avoiding bank conflict and to utilize device memory bandwidth (coalescing read/write) on GPU
Lesson 2: Is It Really Compute-Bound?

- GEMM is n-cube computation
- Bandwidth, bandwidth, bandwidth..., is very important to get the last 10% percentage.
Lesson 3: Programming Model Matters

• No “nice” programming model on CPU yet
• CUDA is relatively easier to programming
• DirectX 11 compute shader provides similar functionality as CUDA, while programming is relative tedious. DirectX 11 on 10 compute shader is limited.
Lesson 4: Memory Model Matters

• Cache-based memory hierarchy provides the flexibility and performance benefits for regular programmer writers.
• Explicit memory access provides experts the similar programming ability.
Challenge: Analytic Model

• Once we have a precise and complete spec, could we derive an analytical model from empirical search to reduce packaging problem at runtime?
Challenge: Design API

• Question: How do we design APIs to provide regular programmers with productivity requirement, and provide experts with efficient API? How much need to be exposed to the expert API?
Conclusion
Microsoft Numeric Library Incubation

Aug. 12, 2009
Charles Fu
Many-Core Library Incubation

• Part of *Parallel Computing Initiative*

• Address many-core parallelism for Microsoft Windows
  – Different from HPC parallelism

• Investigate numerical libraries for many-core architectures

• Collaborations with Robert van de Geijn (FLAME) and Jack Dongarra (PLASMA)
Incubation objectives

• Develop an initial parallel numerical library specifically for Microsoft platforms
• Collaborate on APIs available across platforms
• Demonstrate use of library in parallel applications
• Contribute to adoption of Microsoft platforms on future many-core and HPC architectures
• Linear Algebra is initial focus of the project
Interfaces

• Investigate the design of interfaces
  – “Productivity” interfaces allow users to ignore low level details
  – “Expert” interfaces allow users to access all algorithmic and performance parameters
**Productivity interface example**

**FORTRAN**

*Compute the Cholesky factorization $A = U'*U$

CALL DPOTRF( UPLO, N, A, LDA, INFO )

IF ( INFO.EQ.0 ) THEN

* Solve the system $A*X = B$, overwriting $B$ with $X$.

CALL DPOTRS( UPLO, N, NRHS, A, LDA, B, LDB, INFO )

END IF

**C#**

using MathNS.Array;

// Initialize matrix B and A. A is symmetric positive definite
DoubleArray B = new DoubleMatrix(10,10)
B.FillRandom();
DoubleArray C = new DoubleMatrix(10,10);
C.FillRandom();
DoubleArray A = C * C.T;

Cholesky chol = new Cholesky(A);
if (chol.Succeeded)
    X = chol.Solve(B);
Productivity interface example

**FORTRAN**

*Compute the LU factorization of the matrix A and solve the system A*X = B, overwriting B with X.*

CALL DGESV( N, NRHS, A, LDA, IWORK, B, LDA, INFO )

---

**F#**

```fsharp
open MathNS.Array
open MathNS.MathFunc
let A = randmat(10,10)
let b = randmat(10,1)

let s = Solver(A)
let x = s.Solve(b)

s.MethodUsed
<MathNS.Array.SolverMethod [SolveLU]>
```

**IronPython**

```python
from MathNS.Array import *
from MathNS.MathFunc import *
A = randmat(10,10)
b = randmat(10,1)

s = Solver(A)
x = s.Solve(b)
s.MethodUsed
`<MathNS.Array.SolverMethod [SolveLU]>`
```
Productivity interface example

**FORTRAN**

*Compute \( W = \alpha x + y \)

```fortran
CALL DCOPY (N, Y, INCY, W, INCW)
CALL DAXPY (N, ALPHA, X, INCX, W, INCW)
```

**Native C++**

```cpp
#include <MSBLAS.h>

vector<double> x(10,1.0), y(10,2.0), w;
int alpha=5;

w=alpha*x+y;
```
Algorithms-by-blocks

- Storage by-block format: locality, distribution and parallelism.

- Concept used by HTA, libFLAME (UT Austin), PLASMA (UTK)
C++ Matrix abstractions

• New abstractions for linear algebra
• Allow to create and manage block views of a matrix
C++ Matrix abstractions

• Specification of matrix storage abstracted from algorithm
• Abstractions to specify mapping of matrix storage to distributed memory
Enabling Performance

• Increase parallelism
  – Fine-grain tasks in high level algorithms
• Enhance locality
• Transition to “expert” interfaces for fine tuning
Future work

• Distributed memory parallelism on clusters
• Other matrix types (symmetric, triangular)
• Sparse matrix support