Term Project Proposals

1

Lecture 4

Efficient code - conclusion

2

Outline

- Cache Blocking
- BLAS – A standard API for basic linear algebra operations
- Introducing IT Group Cluster2 and its queuing commands.

3

Cache Blocking

4

Efficient code

- We have introduced several loop transformation techniques and data-type considerations last week.
- Essentially, efficient code =
  - Minimal effort (e.g. move invariants out of loop!)
  - Use cache memory → **locality**
  - Good compiler!
- There are two kinds of locality
  - Spatial locality → continuous memory access
  - Temporal locality → blocking

5

Blocking

- Making efficient use of cache memory
  - Once a data is read, use it as many times as possible (cache-reuse)
- Classical example: matrix-matrix multiplication

6

Matrix-matrix multiplication naïve version

```
typedef unsigned int uint;
void naive(const uint N, stopWatch &t) {
  double a[N][N];
  double b[N][N];
  double c[N][N];
  init(N, &a[0][0], &b[0][0], &c[0][0]);

  for(uint i=0;i<N;i++)
    for(uint j=0;j<N;j++)
      for(uint k=0;k<N;k++)
        c[i][j] += a[i][k] * b[k][j];
}
```

/home/courses/Lecture04/00.cpp

7

Matrix-matrix multiplication with Blocking

```
const uint blk=16; ← need to be tuned empirically

for(uint i0=0;i0<N;i0+=blk)
    for(uint j0=0;j0<N;j0+=blk)
      for(uint k0=0;k0<N;k0+=blk)
        for(uint i=i0;i<min(i0+blk,N);i++)
          for(uint j=j0;j<min(j0+blk,N);j++)
            for(uint k=k0;k<min(k0+blk,N);k++)
              c(i, j) += a(i, k) * b(k, j);
```

/home/courses/Lecture04/00.cpp

8

Reference: http://www.netlib.org/utk/papers/autoblock/node2.html

BLAS

Basic Linear Algebra Subprograms

Introduction to BLAS

Programming with BLAS

9

BLAS (1)

- Basic Linear Algebra Subroutines/Subprograms
  - High quality "building block" routines for performing basic vector and matrix operations
  - Defines three levels of subroutines
    - Level 1: vector-vector operation
      - Norm, scaling, rotation, y ← ax + y
    - Level 2: matrix-vector operation
      - Matrix is stored in general (2-D array), banded, triangular, symmetric, …
    - Level 3: matrix-matrix operation
      - Matrix-matrix multiplication, Triangular solve (after LU decomposition), …

10

BLAS (2)

- Four different data types: S, D, C, Z
- Several matrix types: general, banded, symmetric, …
- A quick reference is available at :
  http://www.netlib.org/blas/blasqr.ps

11

Available BLAS Implementations

- There are several versions of BLAS available, some of the most notable ones:
  - Reference implementation
    http://www.netlib.org/blas/
  - Intel MKL (Math Kernel Library):
    http://www.intel.com/software/products/mkl/
  - ATLAS (Automatically Tuned Linear Algebra Software)
    http://math-atlas.sourceforge.net/
  - ACML (AMD Core Math Library)
    http://developer.amd.com/cpu/Libraries/acml/downloads/pages/default.aspx
  - Kazushige Goto
    http://www.tacc.utexas.edu/resources/software/

12

BLAS (continued)

- Try vendor-specific BLAS libraries first if it is available and affordable.
  - For Linux, Intel MKL is **currently** free for educational and evaluation purposes. (Windows users are not so lucky~)
  - Vendors should know their processor/computer best, thus it should be the best performing one to use.
- Try ATLAS, which automatically tunes itself during compilation.
  - Be warned it is a long process!
  - ATLAS sometimes beats vendor implementations
- As ATLAS is free and of high-quality, reference implementation on netlib really is for reference.

How to use BLAS?

- We've got Intel MKL installed in our cluster
  - Version: 8.0, 8.1, 9.0, 9.1,10.0, 10.1, 10.2, 10.3
  - /opt/intel/mkl/current (10.2)
  - Documentation: /opt/intel/mkl/current/doc
  - Manual: /opt/intel/mkl/current/doc/mklman.pdf
    - http://140.118.105.174/docs/IntelMKL/mklman.pdf
  - User guide:
    - http://140.118.105.174/docs/IntelMKL/userguide.pdf
  - Examples: /opt/intel/mkl/current/examples

BLAS-1

- Level 1: 01_BLAS-1.c

```
cblas_dnrm2(const int size,
  double *vec, const int incX)

cblas_ddot(const int size,
  double *vecX, const int incX,
  double *vecY, const int incY)
```

  - With good compiler (e.g. Intel), the performance of BLAS-1 in MKL should not be very different from the vanilla C code.
  - We're using CBLAS here! (C binding for BLAS)

Example: 01_BLAS-1.cpp

```
#include <iostream>
#include "mkl.h"
using namespace std;

int main() {
    double a[]={1, 2, 3, 4, 5};
    double b[]={0, 0, 2, 3, 1};

    cout << "\nInner dot: " << cblas_ddot(5, a, 1, b, 1);
    cout << "\nLength of a: " << cblas_dnrm2(5, a, 1);
    cout << "\nLength of b: " << cblas_dnrm2(5, b, 1);

    return 0;
}
```

```
Inner dot: 23.000000
Length of a: 7.416198
Length of b: 3.741657
```


Example: 02_BLAS-1-Fortran.cpp

```
#include <iostream>
#include "mkl.h"
using namespace std;

int main() {
    double a[]={1, 2, 3, 4, 5};
    double b[]={0, 0, 2, 3, 1};
    int n=5, inc=1;

    cout << "\nInner dot: " << ddot(&n, a, &inc, b, &inc);
    cout << "\nLength of a: " << dnrm2(&n, a, &inc);
    cout << "\nLength of b: " << dnrm2(&n, b, &inc);
    return 0;
}
```

```
Inner dot: 23.000000
Length of a: 7.416198
Length of b: 3.741657
```


How to use BLAS? (continued)

- BLAS-1: in BLAS, vectors are defined with an increment to ensure maximum flexibility.
  - The increment defines the "distance" between two entries in a vector.
  - Column vectors (in C) has an increment of nCol (number of columns)
  - Row vectors (in Fortran) has an increment of nRow (number of rows)

- All BLAS subroutines have names start with the variable type: s, d, c, z – single precision, double precision, single precision complex, double precision complex.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

C: Row vector, increment=1
C: Column vector: increment=5

1	5	9	13	17
2	6	10	14	18
3	7	11	15	19
4	8	12	16	20

Fortran: Row vector, increment=4
Fortran: Column vector: increment=1

19

Example: 03_BLAS-2.cpp

```
#include <iostream>
#include "mkl.h"
using namespace std;

int main() {
    double a[]={1, 2, 3, 4, 5, 6, 7, 8, 9};
    double b[]={0, 0, 2};
    double y[3];
    int i;

    cblas_dgemv(CblasRowMajor, CblasNoTrans, 3, 3,
      1.0, a, 3,       b, 1,     0.0, y, 1);
    cout << "\n";
    for(i=0;i<3;i++) {
        cout << "\n" << y[i];
    }

    return 0;
}
```

20

How to use BLAS? (continued)

- Level 2: 03_BLAS-2.cpp
  - Matrix-vector operations

```
cblas_dgemv([ORDER], [TRANS], m, n,
  alpha, A, lda,
  x, incX,
  beta, y, incY);
```

  - y = alpha * **A** * **x** + beta * **y**
  - For BLAS-2, the next two characters after variable type in function names indicate the matrix type: GE, GB, HE, HB, …
  - For matrices, need to specify transposition, dimensions (m, n), leading dimension (how many entries in the orientation of the matrix)

21

How to use BLAS? (continued)

- Level 3: 04_BLAS-3.cpp
  - Matrix-matrix product

```
cblas_dgemm(CblasRowMajor,
     CblasNoTrans, CblasNoTrans,
   [m], [n], [k],
   [alpha], A, [lda],
   B, [ldb],
   [beta], C, [ldc]);
}
A: [m] x [k]
B: [k] x [n]
C: [m] x [n]
C = alpha * A * B + beta * c
```

22

Example: 04_BLAS-3.cpp

```
#include <iostream>
#include "mkl.h"

int main() {
    double a[]={1, 2, 3, 4, 5, 6};           // 2x3
    double b[]={1, 0, 0, 0, 0, 1};           // 3x2
    double c[4];                             // 2x2
    int i;

    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
      2, 2, 3, 1.0, a, 3, b, 2, 0.0, c, 2);
    for(i=0;i<4;i++) {
        if(i%2==0) std::cout << "\n";
        std::cout << c[i];
    }

    return 0;
}
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 4 & 6 \end{bmatrix}$$

23

Compilation & Linking

- Need to include related header files, and link against the libraries.
  - Command line options for Intel MKL:
    - -I /opt/Intel/mkl/current/include
    - Statically linked: *-static -Wl,--start-group -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -Wl,--end-group*
    - Dynamically linked: *-lmkl_intel_mkl -lmkl_core -liomp5*
  - More details can be found in the manual of MKL
    - http://140.118.105.174/docs/IntelMKL/userguide.pdf
    - Chapter 3: Intel Math Kernel Library Structure
    - Chapter 5: Linking Your Application with Intel Math Kernel Library
    - Chapter 6: Managing Performance and Memory

24

LAPACK

- In additional to BLAS/CBLAS, Intel MKL also includes LAPACK
- LAPACK: **L**inear **A**lgebra **PACK**age
  - Solving systems of linear equations and
  - linear least squares,
  - eigenvalue problems,
  - singular value decomposition.
  - It also includes routines to implement the associated matrix factorizations such as LU, QR, Cholesky and Schur decomposition.

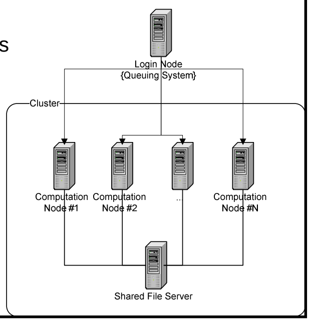
Summary

- BLAS defines three levels of operations:
  - Level 1: vector operations
  - Level 2: matrix-vector operations
  - Level 3: matrix-matrix operations
- Higher level of BLAS subroutines yield better performance due to data access pattern can be rearranged to take advantage of memory hierarchy.
- We should use BLAS whenever possible unless the matrix/vector is small that the overhead of calling subroutines overshadows the benefit of highly tuned subroutines.

IT Group Cluster2

What is a cluster?

- A cluster is a dedicated resource for running computational tasks.
  - A collection of computers



IT Group Cluster2 (1/2)

- Base system: Gentoo Linux (http://www.gentoo.org)
  - 7 Pentium D930 (Dual core, 3.0GHz) nodes
  - 3 Core 2 dual E6320 (dual core, 1.86GHz) nodes
  - All equipped with 4GB RAM (available memory varies)
- Queuing system: SLURM (http://www.llnl.gov/linux/slurm/)
  - 2 public queues (Public, Core) with 30-minute limits
  - 1 private queue without time limit
  - First come, first serve.
- Monitoring system: Ganglia (http://ganglia.sourceforge.net/)
  - http://140.118.5.6:8000

IT Group Cluster2 (2/2)

- **Programming environment**
  - Compilers
    - GNU Compiler Collection suite (gcc) / gcc, g++
    - Intel C/C++/Fortran Compilers
    - *PathScale, PGI*
  - Auxiliary utilities
    - make, gdb, valgrind, gprof, …
  - Supporting programming libraries
    - MPI: OpenMPI, MPICH2
    - Numerical libraries: Intel MKL, AMD ACML, …

Important Commands / Queuing

- **sinfo** → get information about the queue

```
ymhsieh@n00 ~ $ sinfo
PARTITION AVAIL  TIMELIMIT NODES  STATE NODELIST
Public*      up      30:00     8   idle n[00-07]
Core         up      30:00     4   idle c[00-03]
```

```
ymhsieh@n00 ~ $ sinfo
PARTITION AVAIL  TIMELIMIT NODES  STATE NODELIST
Public*      up      30:00     4  alloc n[01-04]
Public*      up      30:00     4   idle n[00,05-07]
Core         up      30:00     4   idle c[00-03]
```

alloc: allocated / occupied

31

Important Commands / Queuing

- **squeue** → information about the queue

```
ymhsieh@n00 ~/Work/04_Test/MPI $ squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES
NODELIST(REASON)
   1008    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1009    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1010    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1006    Public  startup  ymhsieh   R       0:04      3 n[01-03]
   1007    Public  startup  ymhsieh   R       0:03      3 n[04-06]
```

```
PD: Pending
R: Runing
```

32

Important Commands / Queuing

- **scancel** → cancel a job in the queue

```
ymhsieh@n00 ~/Work/04_Test/MPI $ scancel 1011
ymhsieh@n00 ~/Work/04_Test/MPI $ squeue
  JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON
   1010    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1012    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1013    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1014    Public  startup  ymhsieh  PD       0:00      3 (Resources)
   1008    Public  startup  ymhsieh   R       0:38      3 n[01-03]
   1009    Public  startup  ymhsieh   R       0:37      3 n[04-06]
```

33

Important Commands / Queuing

- **sbatch** (batch processing) → submit a *job script* for later execution. The script will typically contain one or more srun commands to launch parallel tasks.
  - sbatch –n8 /opt/mpich2/startup ./myProg.exe
- **srun** (interactive processing) → submit a job for execution or initiate job steps in interactive mode.
  - srun ./myProg.exe
  - srun -n8 ./myProg.exe
  - srun -n8 ./myProg.exe 200
- Complete Reference: https://computing.llnl.gov/linux/slurm/quickstart.html

34

Summary

- sinfo
- squeue
- scancel
- sbatch
- srun

35

Assignment #3

36