High Performance Computing for Science and Engineering I

18.10.2019 - HPC Basics - Tutorial II - BLAS

Dr. Sergio Martin

CSElab

Computational Science & Engineering Laboratory

ETH zürich

BLAS (Basic Linear Algebra Subprograms)

BLAS is a set of routines that provide standard building blocks for performing basic vector and matrix operations

Level 1: O(n) Vector Operations

- DROTG - setup Givens rotation
- DROTMG - setup modified Givens rotation
- DROT - apply Givens rotation
- DROTM - apply modified Givens rotation
- DSWAP - swap x and y
- DSCAL - x = a*x
- DCOPY - copy x into y
- DAXPY - y = a*x + y
- DDOT - dot product
- DSDOT - dot product with extended precision accumulation
- DNRM2 - Euclidean norm
- DZNRM2 - Euclidean norm
- DASUM - sum of absolute values
- IDAMAX - index of max abs value

Level 2: O(n^2) Vector/Matrix Operations

- DGEMV - matrix vector multiply
- DGBMV - banded matrix vector multiply
- DSYMV - symmetric matrix vector multiply
- DSBMV - symmetric banded matrix vector multiply
- DSPMV - symmetric packed matrix vector multiply
- DTRMV - triangular matrix vector multiply
- DTBMV - triangular banded matrix vector multiply
- DTPMV - triangular packed matrix vector multiply
- DTRSV - solving triangular matrix problems
- DTBSV - solving triangular banded matrix problems
- DTPSV - solving triangular packed matrix problems
- DGER - performs the rank 1 operation A := alpha*x*y' + A
- DSYR - performs the symmetric rank 1 operation A := alpha*x*x' + A
- DSPR - symmetric packed rank 1 operation A := alpha*x*x' + A
- DSYR2 - performs the symmetric rank 2 operation, A := alpha*x*y' + alpha*y*x' + A
- DSPR2 - performs the symmetric packed rank 2 operation, A := alpha*x*y' + alpha*y*x' + A

Level 3: O(n^3) Matrix Operations

- DGEMM - matrix matrix multiply
- DSYMM - symmetric matrix matrix multiply
- DSYRK - symmetric rank-k update to a matrix
- DSYR2K - symmetric rank-2k update to a matrix
- DTRMM - triangular matrix matrix multiply
- DTRSM - solving triangular matrix with multiple right hand sides

Open in your browser:
http://www.netlib.org/blas/

Installing CBLAS (as included in LAPACK)

If you are using Euler (or any other cluster), you can use a pre-compiled module: $*module load lapack*

If you are using Linux (e.g., Ubuntu), you can probably download it: *$sudo apt-get install lapack*

However, we are now going to download and install LAPACK manually.

Step 1) Create and go to a new folder:
$ mkdir ~/src
$ cd ~/src

Step 2) Download BLAS:
$ wget http://www.netlib.org/lapack/lapack-3.8.0.tar.gz

Step 3) Unpack tar file:
$ tar -xzvf lapack-3.8.0.tar.gz

Step 4) Go to the blas source folder:
$ cd lapack-3.8.0/

Step 5) Create makefile configuration and Make:
$ cp make.inc.example make.inc; **make blaslib cblaslib -j4**

Step 6) Copy libcblas.a and librefblas.a to the tutorial folder. $ cp *.a ~/tutorial2

Practicum I:

Use CBLAS To Calculate the Dot Product of two Vectors

Code:

p1.cpp

CBLAS Routines Reference:

https://www.gnu.org/software/gsl/doc/html/cblas.html

Practicum II:

Let’s improve our own dot product...

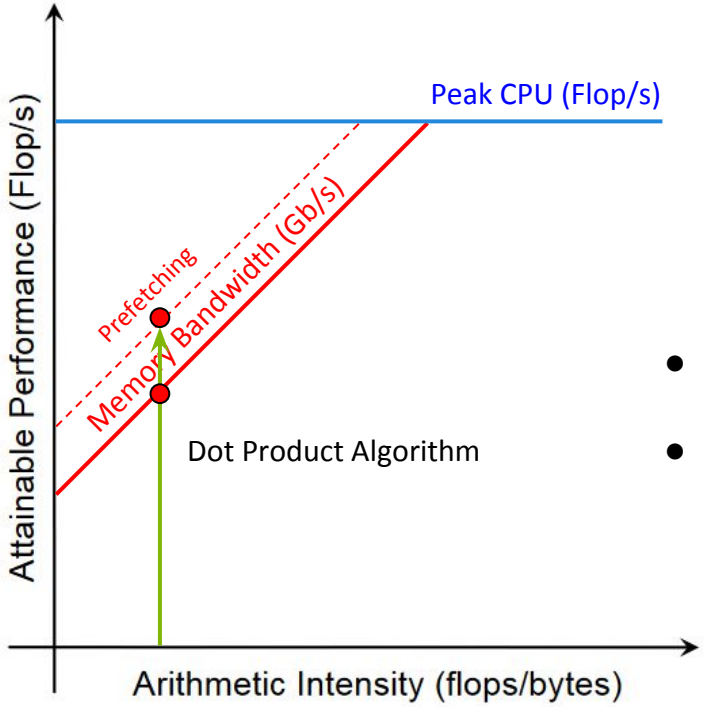
Code:

p1.cpp

CBLAS Routines Reference:

https://www.gnu.org/software/gsl/doc/html/cblas.html

Effect of Loop Unrolling



- But also reduces loop overhead
- And enables vectorization (we discuss this later)

Practicum III:

Use CBLAS To Calculate the Matrix/Vector Multiplication

Code:

p2.cpp

CBLAS Routines Reference:

https://www.gnu.org/software/gsl/doc/html/cblas.html

Hints: Matrix Majors: CblasRowMajor, CblasColMajor

Matrix Transposing: CblasNoTrans, CblasTrans

LDA: Leading Dimension

Practicum IV:

Improve the Performance of the Manual Variant.

Code:

p2.cpp

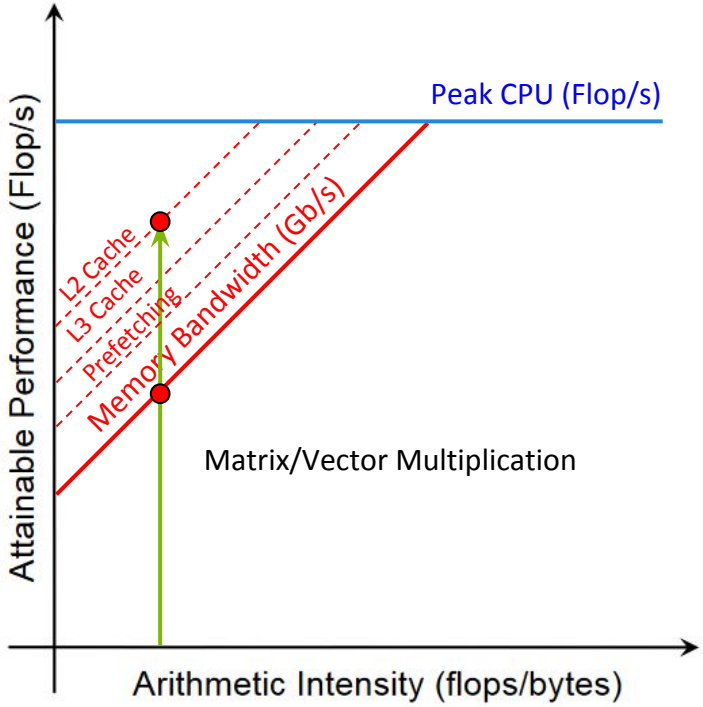
CBLAS Routines Reference:

https://www.gnu.org/software/gsl/doc/html/cblas.html

Hints: Matrix Majors: CblasRowMajor, CblasColMajor
Matrix Transposing: CblasNoTrans, CblasTrans
LDA: Leading Dimension

Effect of Effective Cache Use

Roofline Model: Predicts the performance of an application on a given multi-core node.



Source: S. Williams et al. "*Roofline: An Insightful Visual Performance Model for Multicore Architectures*"

Practicum V:

Use CBLAS to compute Matrix-Matrix Multiplication

Code:

p3.cpp

CBLAS Routines Reference:

https://www.gnu.org/software/gsl/doc/html/cblas.html

Hints: Matrix Majors: CblasRowMajor, CblasColMajor
Matrix Transposing: CblasNoTrans, CblasTrans
LDA: Leading Dimension

Practicum VI:

Use Cache-blocking to optimize Matrix/Matrix Multiplication

Code:

p3.cpp

CBLAS Routines Reference:

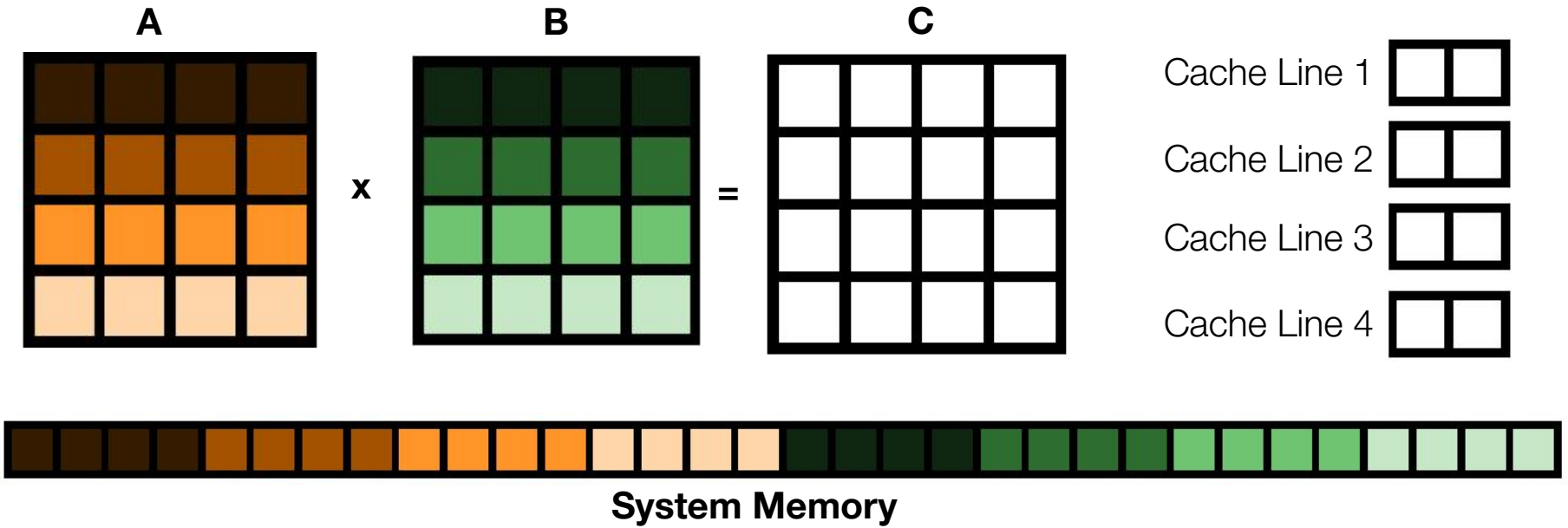
https://www.gnu.org/software/gsl/doc/html/cblas.html

Hints: Matrix Majors: CblasRowMajor, CblasColMajor
Matrix Transposing: CblasNoTrans, CblasTrans
LDA: Leading Dimension

Cache Usage: Matrix Matrix Multiplication

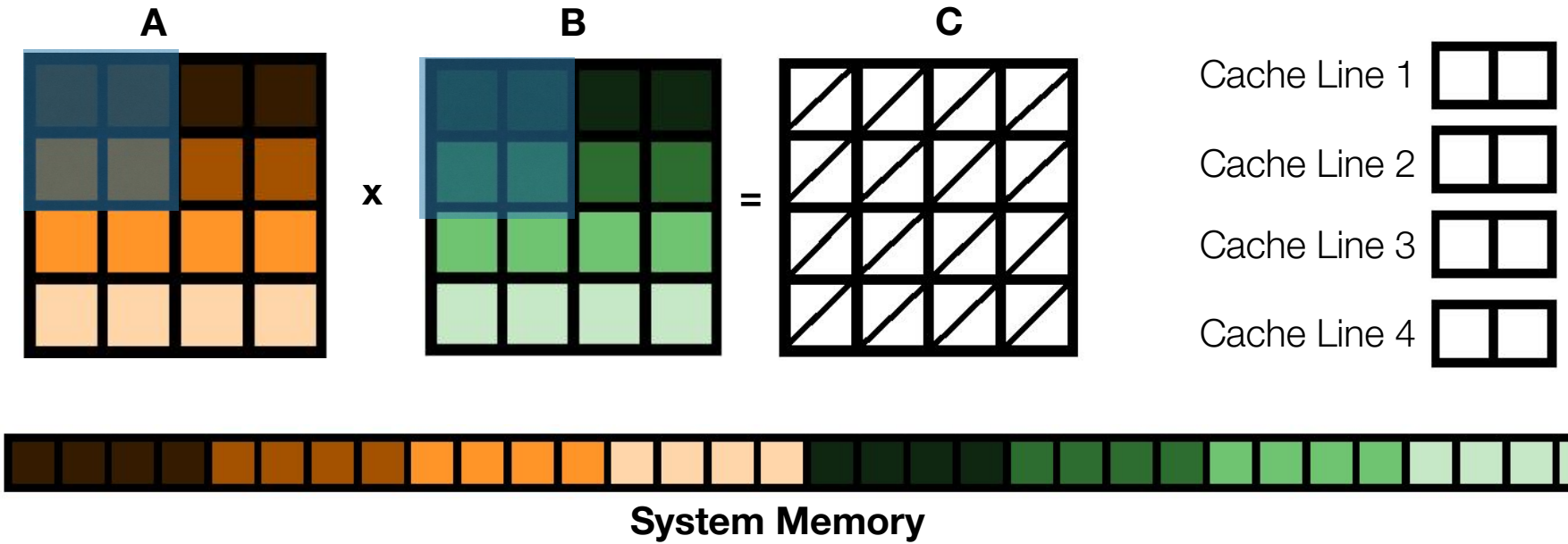
A and B stored row-major differentiated color. Let's ignore C accesses for simplicity.

4 Cache Lines of 2 Elements Each

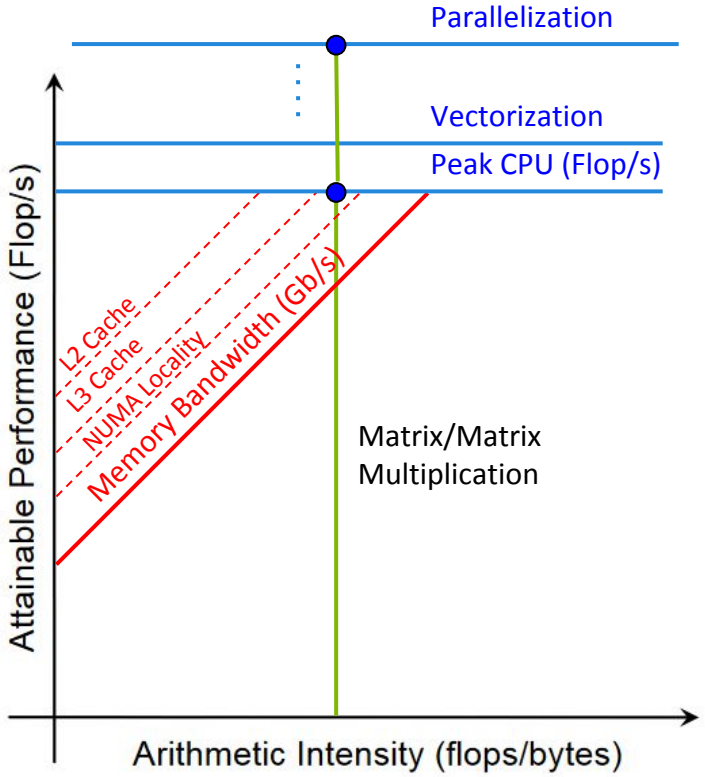


Cache Optimization: Blocked Multiplication

Idea: Let's solve the multiplication in blocks, storing partial solutions to $C_{i,j}$



We Reached Peak CPU, What Now?

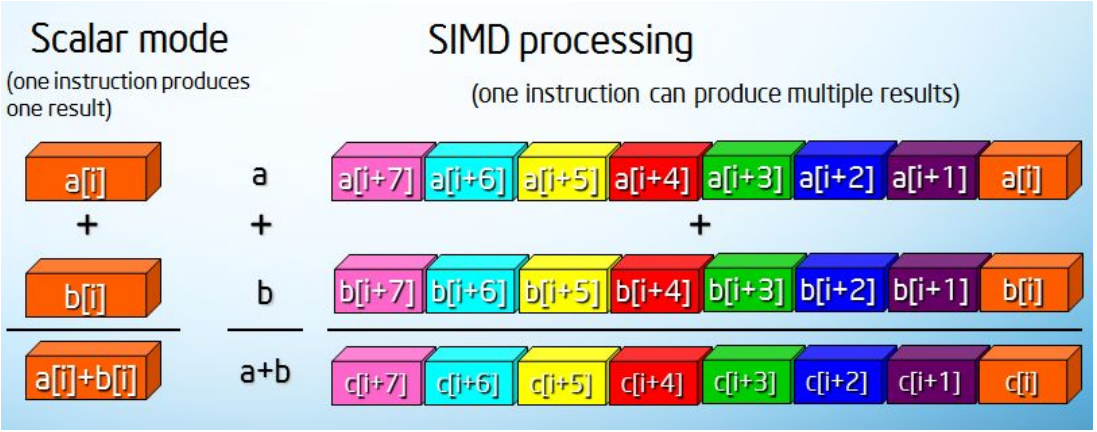


Source: S. Williams et al. "*Roofline: An Insightful Visual Performance Model for Multicore Architectures*"

Vectorization

Single Instruction, Multiple Data Paradigm

Apply the same instruction to an array of elements, instead one-by-one.



Source: Vectorization - Find out what it is, Find out More!, Shannon Cepeda (Intel)

Three ways to go about it:

1. Compiler-Enabled Auto-Vectorization (GNU or Intel). (Easiest option - It's ok)
2. Intel SPMD Program Compiler (ISPC)
3. Intel Intrinsics

Practicum VII:

Use OpenMP to parallelize your Matrix/Matrix Multiplication Algorithm

Code:

p3.cpp

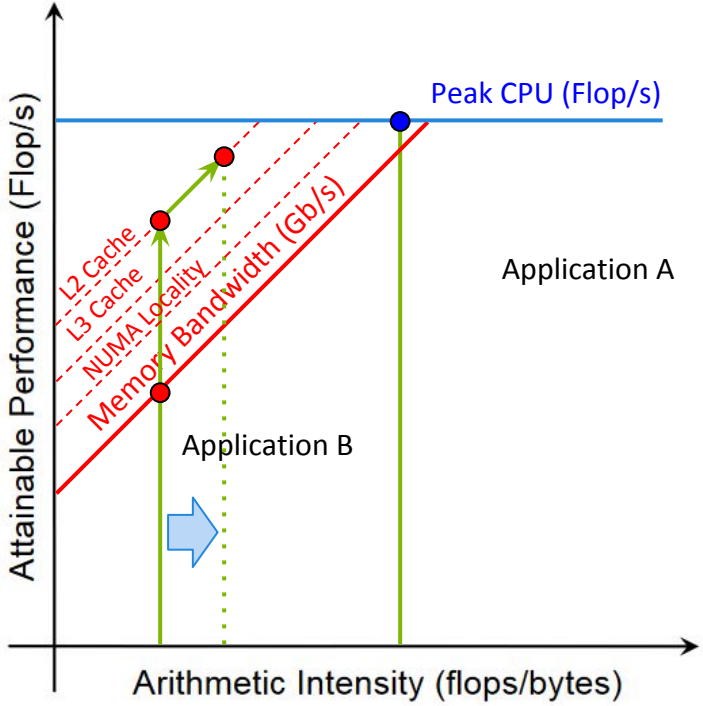
CBLAS Routines Reference:

https://www.gnu.org/software/gsl/doc/html/cblas.html

Hints: Matrix Majors: CblasRowMajor, CblasColMajor
Matrix Transposing: CblasNoTrans, CblasTrans
LDA: Leading Dimension

Cost of Internal Communication (II)

Roofline Model: Predicts the performance of an application on a given multi-core node.



Example Applications:

A) Matrix Multiplication (High Arithmetic Intensity)

B) Structured Grid Solver (Low Arithmetic Intensity)

Optimizations

- Optimize for NUMA/Cache Locality (e.g. Cache Blocking)
- Increase Arithmetic Intensity (e.g. Prevent unnecessary data motion)

Parallel Programming Models play a key role in this!

Source: S. Williams et al. "*Roofline: An Insightful Visual Performance Model for Multicore Architectures*"