CS141: Deep Learning & Optimizing Matrix Multiplication

CS 141 Staff

Outline

Modern Deep Learning Frameworks

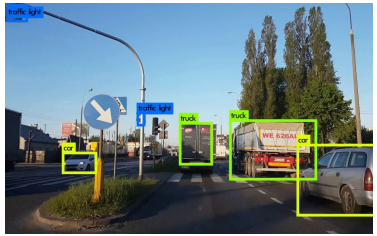
Matrix Multiplication

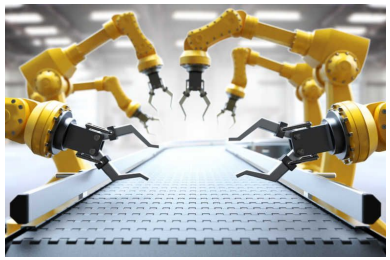
Optimizing Matrix Multiplication

Final Project Overview

Deep Learning Infrastructure

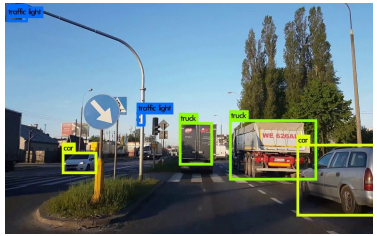
Applications

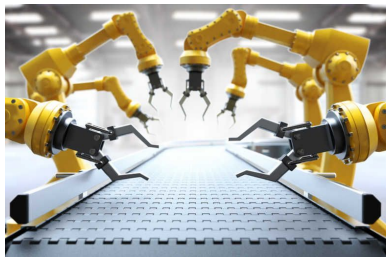




Deep Learning Infrastructure

Applications





Tools

TensorFlow

Caffe2

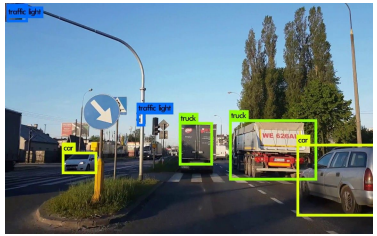
Keras

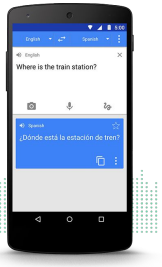
PYTORCH

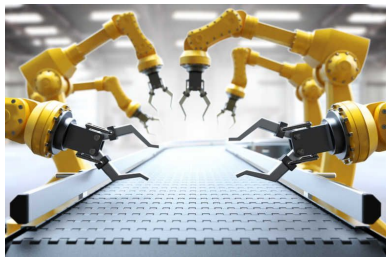
mxnet

Deep Learning Infrastructure

Applications







Tools

TensorFlow

Caffe2

Keras

PYTORCH

mxnet

Backend

Eigen

intel MKL Library

cuBLAS

cuDNN

Modern Deep Learning Frameworks

Eigen

MKL Library

cuBLAS

cuDNN

Linear Algebra

Modern Deep Learning Frameworks

Eigen

intel MKL Library

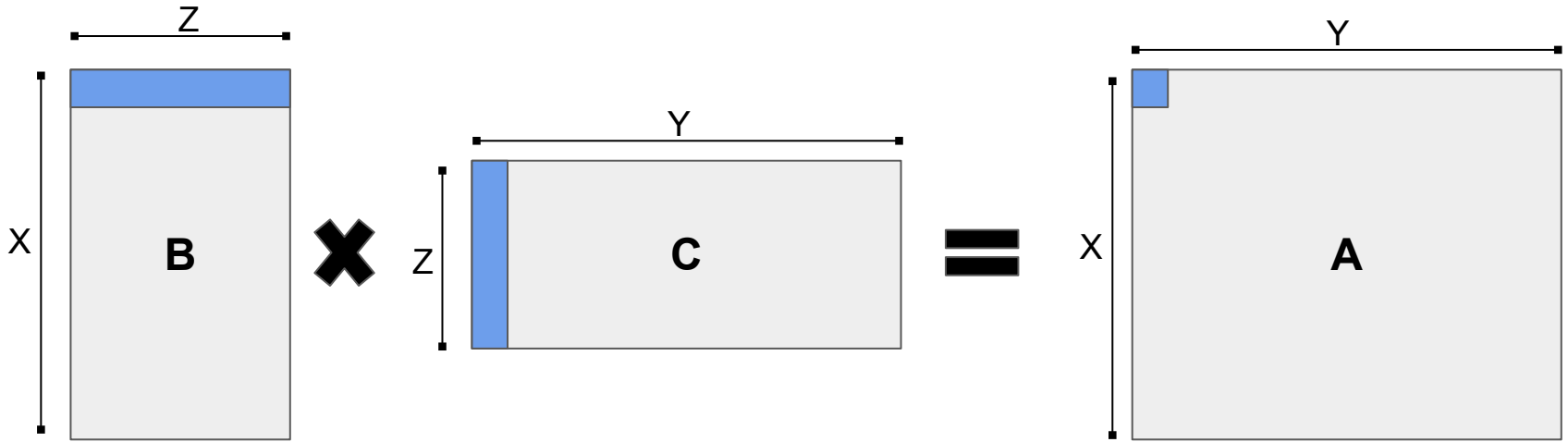
cuBLAS

cuDNN

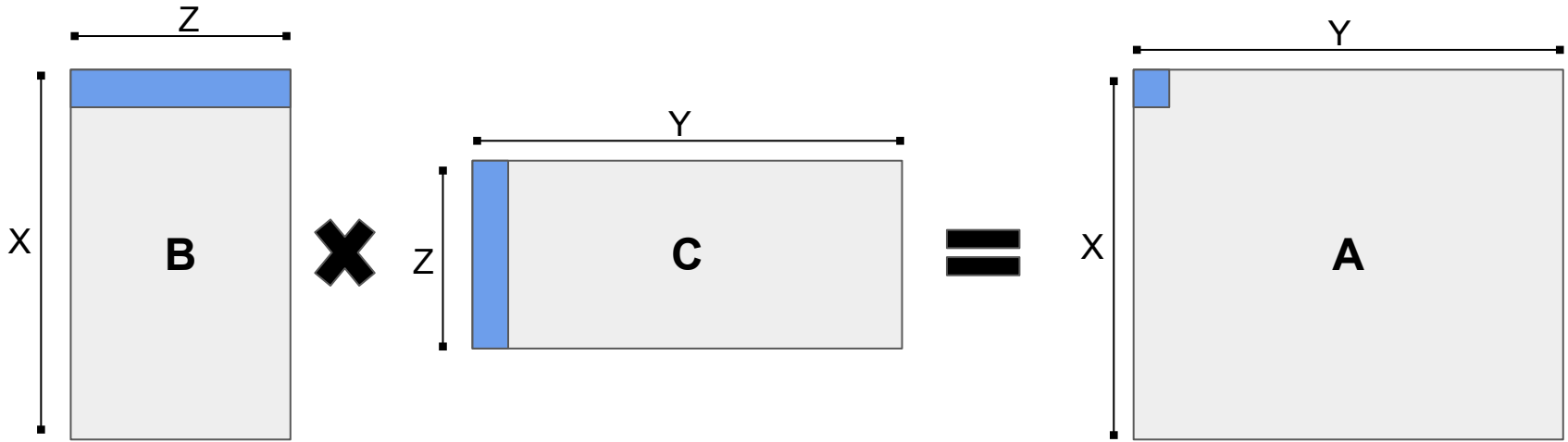
Linear Algebra

Matrix Multiplication

Matrix Multiplication, BC = A

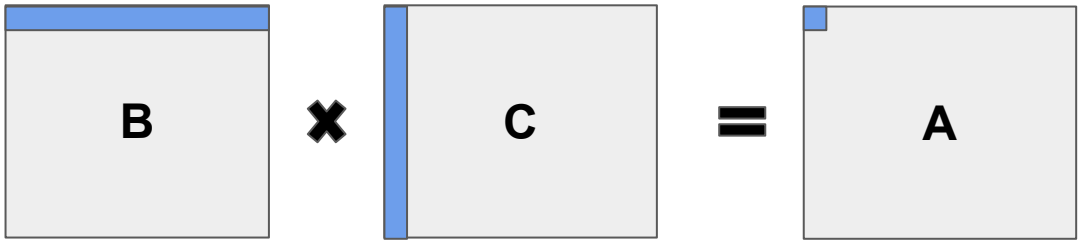


Matrix Multiplication, BC = A



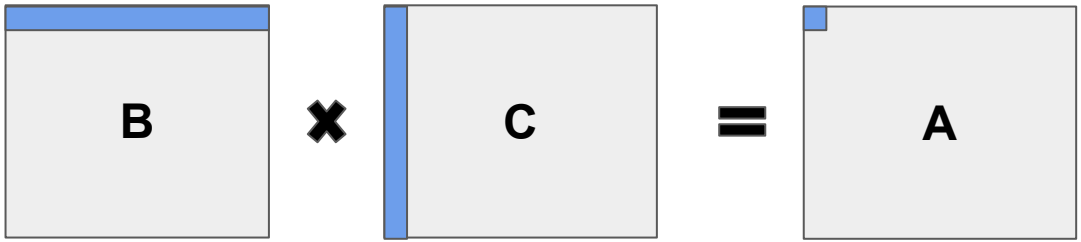

```
for(int i =0; i < x; i++)
        for(int j =0; j < y; j++)
                for(int k=0; k < z; k++)
                        A[i][j] += B[i][k]*C[k][j]
```

Matrix Multiplication: Example



- Let’s assume B and C are both 4096 x 4096 (4K x 4K)
  - **How big are each of these matrices (assume 4 Byte per element)?**

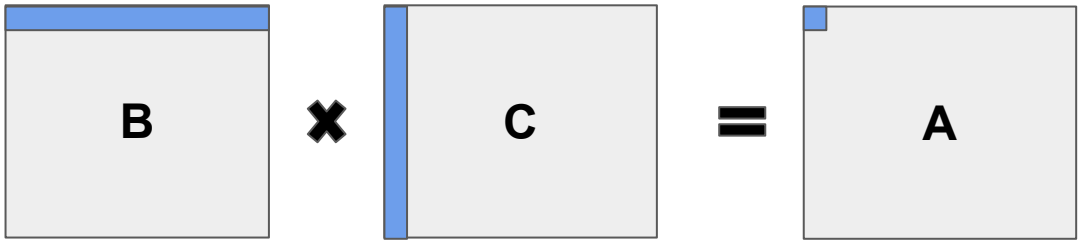
Matrix Multiplication: Example



64MB 64MB 64MB

- Let's assume B and C are both 4096 x 4096 (4K x 4K)
  - **How big are each of these matrices (assume 4 Byte per element)?**

Matrix Multiplication: Example



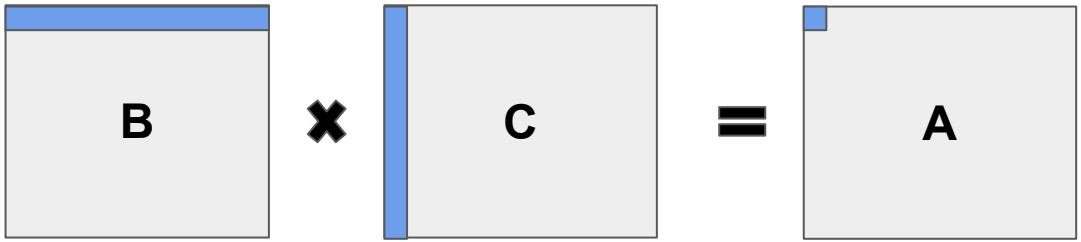

```
for(int i =0; i < x; i++)
        for(int j =0; j < y; j++)
                for(int k=0; k < z; k++)
                        A[i][j] += B[i][k]*C[k][j]
```

64MB 64MB 64MB

- Let's assume B and C are both 4096 x 4096 (4K x 4K)
  - **With a naive implementation, how long will this take to run?**

(a) Milli-seconds
(b) Seconds
(c) Minutes
(d) Hours
(e) Days

Matrix Multiplication: Example



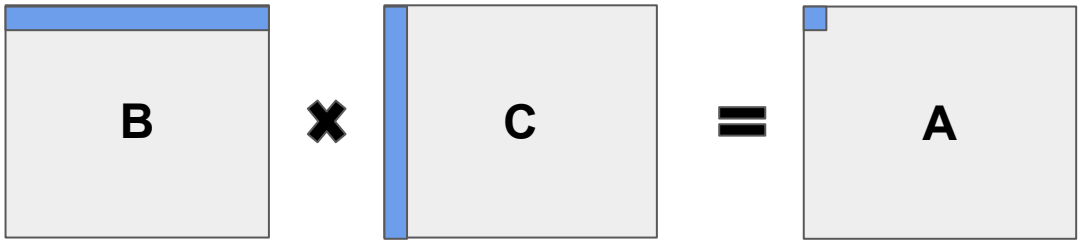

```
for(int i =0; i < x; i++)
        for(int j =0; j < y; j++)
                for(int k=0; k < z; k++)
                        A[i][j] += B[i][k]*C[k][j]
```

64MB 64MB 64MB

- Let's assume B and C are both 4096 x 4096 (4K x 4K)
  - **With a naive implementation, how long will this take to run?**

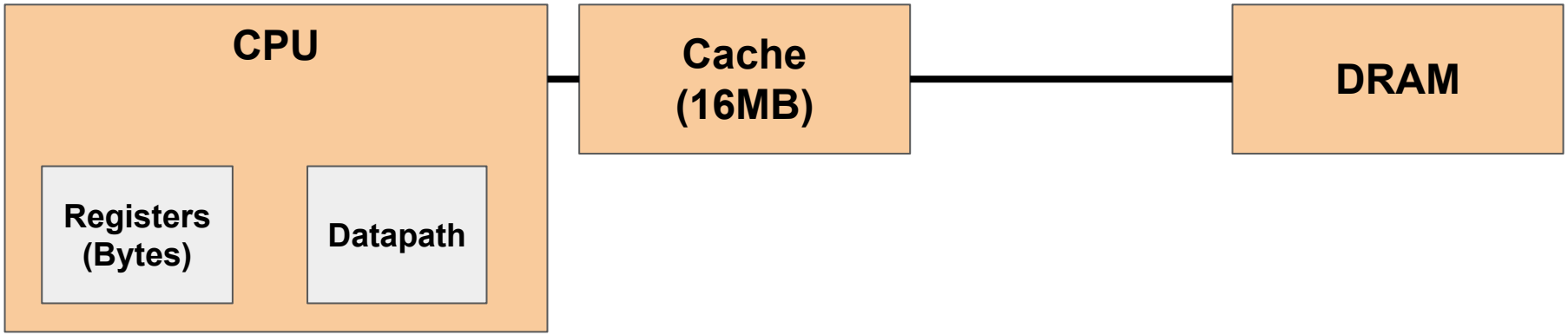
(a) Milli-seconds
(b) Seconds
(c) Minutes
(d) Hours
(e) Days

Matrix Multiplication: Why so slow?



64MB 64MB 64MB

```
for(int i =0; i < x; i++)
        for(int j =0; j < y; j++)
                for(int k=0; k < z; k++)
                        A[i][j] += B[i][k]*C[k][j]
```



Naive Matrix Multiplication Implementation

How many slow memory fetches do we actually have?

```
for(int i =0; i < N; i++)
        for(int j =0; j < N; j++)
                for(int k=0; k < N; k++)
                        A[i][j] += B[i][k]*C[k][j]
```

(Hint: How many times we do fetch each element in C?)

Naive Matrix Multiplication Implementation

- How many slow memory fetches do we actually have?

```
for(int i =0; i < N; i++)
        [read row i of B into fast memory]
        for(int j =0; j < N; j++)
                [read A(i,j) into fast memory]
                [read column j of C into fast memory]
                for(int k=0; k < N; k++)

                        A[i][j] += B[i][k]*C[k][j]
                [write A(i,j) back to slow memory]
```

Total =

N^3 + (C)

N^2 + (B)

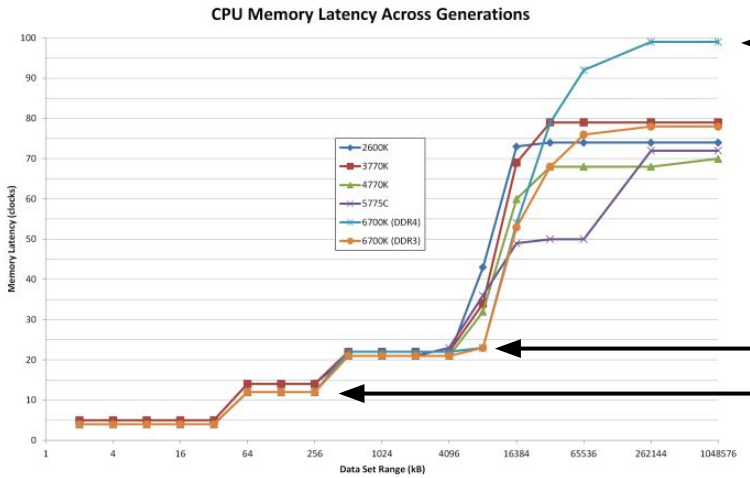
2N^2 (Write & Read A)

Total = N^3 + 3N^2

Total ~= 68 Billion

Naive Matrix Multiplication Implementation

Do 68 billion memory fetches from DRAM really take hours?



DRAM Memory Latency

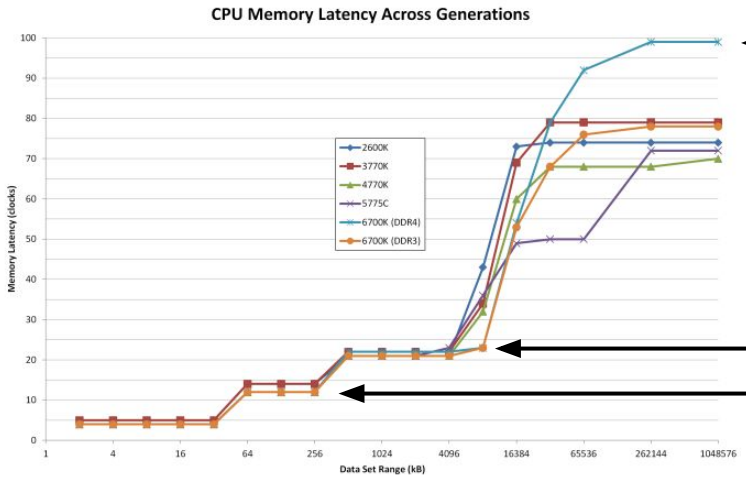
L2 Cache Memory Latency

L1 Cache Memory Latency

68 Billion DRAM Fetches * (100 cycles / DRAM fetch) * (1 / (2 GHz)) = 1 hour

Naive Matrix Multiplication Implementation

Do 68 billion memory fetches from DRAM really take hours?



DRAM Memory Latency

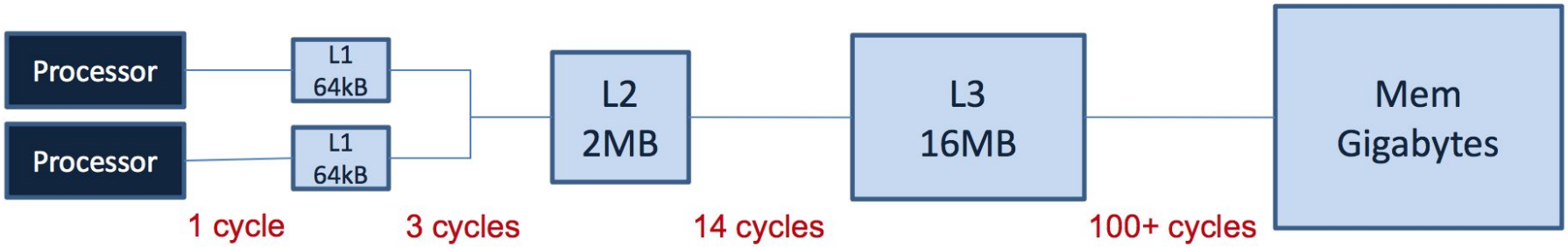
L2 Cache Memory Latency

L1 Cache Memory Latency

68 Billion DRAM Fetches * (100 cycles / DRAM fetch) * (1 / (2 GHz)) = 1 hour

How can we do better?

Optimizing Matrix Multiplication



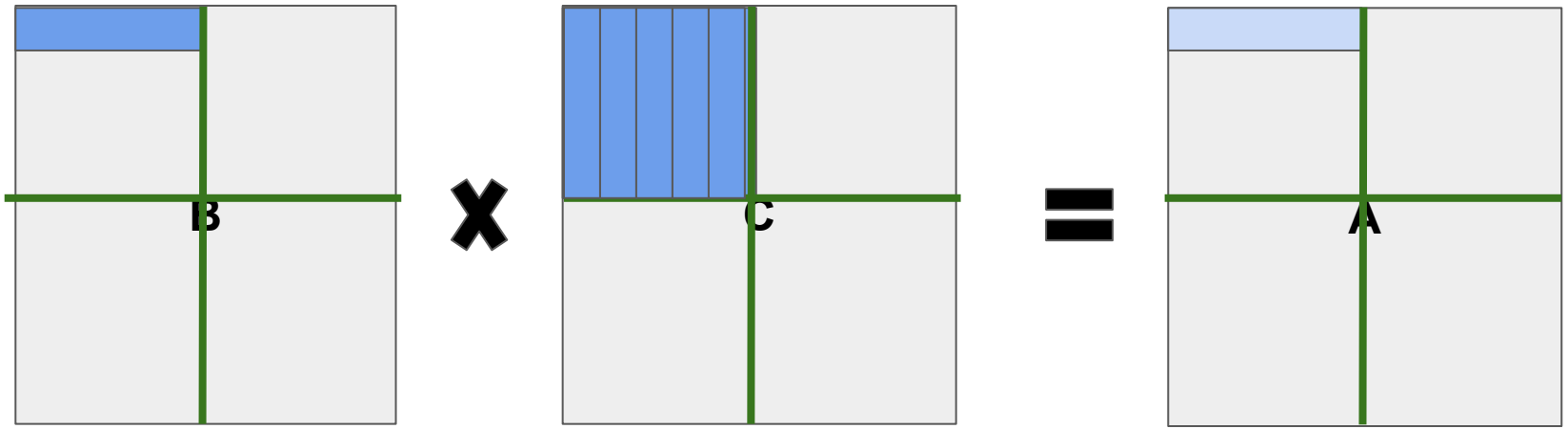
- Let's try to impose and improve **locality**

- Caches provide **faster,** but **limited storage**
  - How can we improve software to reuse cached values?

Optimizing Matrix Multiplication: Tiling

Goal: Reduce number of slow memory fetches.

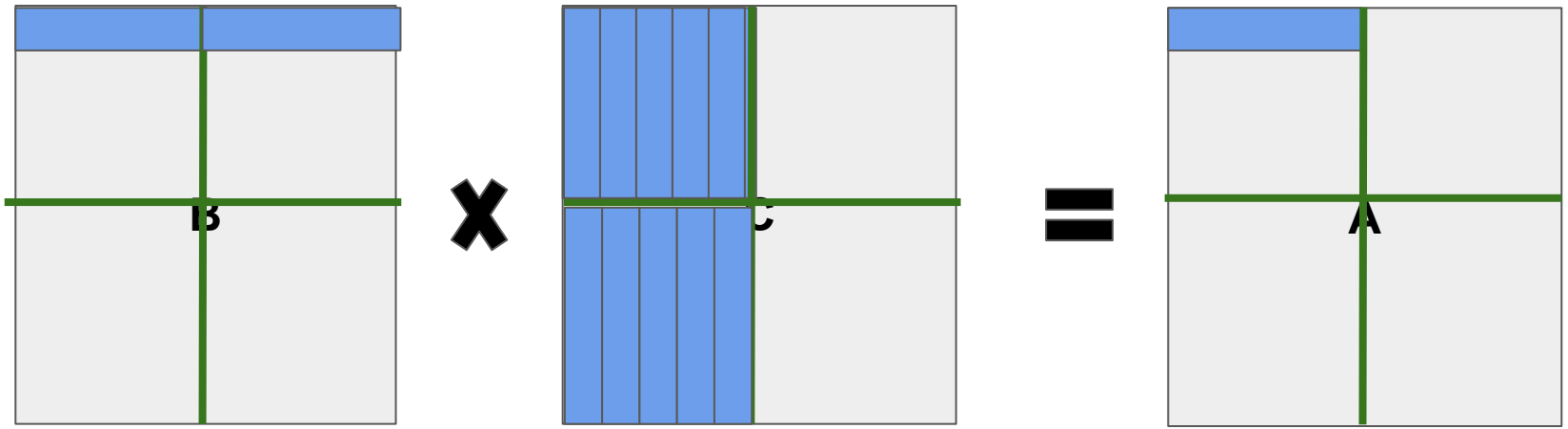
Solution: Reuse data in caches



Optimizing Matrix Multiplication: Tiling

Goal: Reduce number of slow memory fetches.

Solution: Reuse data in caches



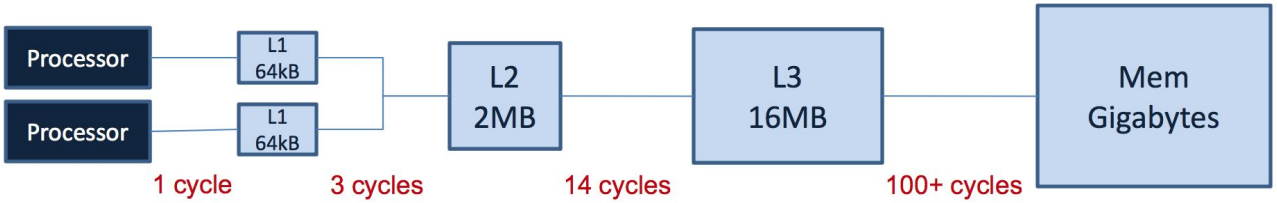
Optimizing Matrix Multiplication: Tiling

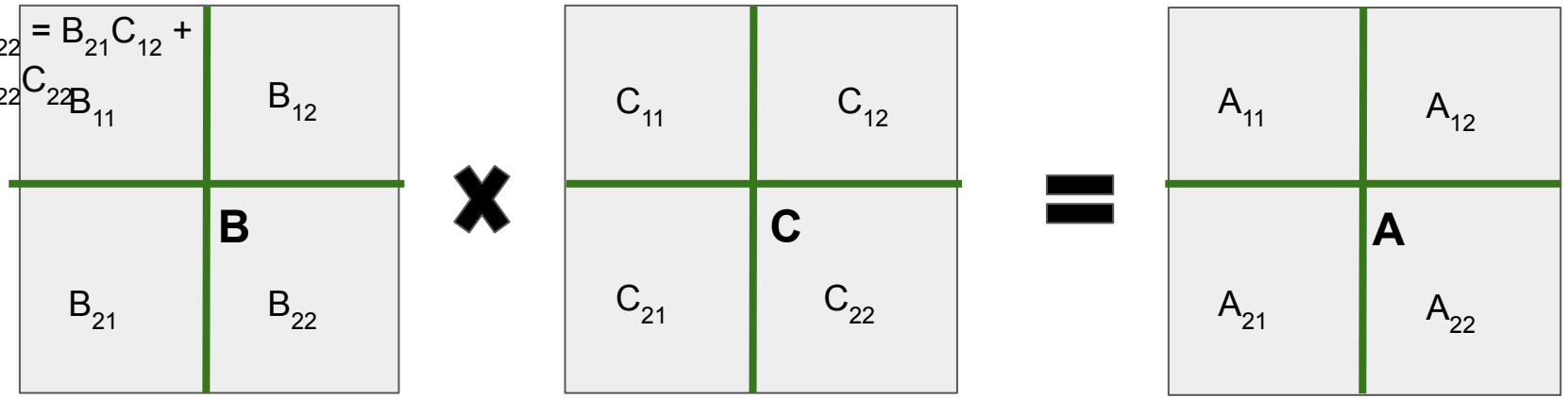
$A_{11} = B_{11}C_{11} + B_{12}C_{21}$

$A_{12} = B_{11}C_{12} + B_{12}C_{22}$

$A_{21} = B_{21}C_{11} + B_{22}C_{21}$

$A_{22} = B_{21}C_{12} + B_{22}C_{22}$





Final Project Overview

1. Modifying your MIPS processor

- Support for mul instruction
- Assembly for 8x8 matrix multiplication

2. Software Optimization of Matrix Multiplication

- Implementation of tiled matrix multiplication (in C)
- Vary tile size, measure execution time, compare to naive implementation

3. Written Report

- Motivate optimizations and interpret your results
- Include tables, plots, and written analysis

4. Extra Credit

Final Project Extra Credit

Extending and implementing additional software optimizations.

- Implement an alternative optimization or modify/extend your tiled implementation
- Measure and report execution time for your alternative optimization compared to the naive implementation and your tiled implementation
- Interpret results and include summary and analysis in your written report
    - Provide and explain an example of when your optimization makes sense (i.e., depending on matrix size or other properties).
- We will provide several examples of additional optimizations on Canvas, but you are encouraged to think about your own alternatives

Final Project Logistics

Deliverables:

1. Project file for modified version of processor
2. zip file of all source code for your matrix multiplication implementation & related scripts
3. pdf of written report

Due Date: May 8th, 2019 at 11:59pm

Collaboration: *Individual* project but can work with partner on the modified processor *only*