Optimization 101

Christopher Blanton

PACE

Summer 2020

Introduction

Motivation

- ***Maximizing computational results while minimizing the amount of computational resources is code optimization.***

Motivation

- ***Maximizing computational results while minimizing the amount of computational resources is code optimization.***
- ***There are many reasons why we want to do this, but here are two common ones:***

Motivation

- ***Maximizing computational results while minimizing the amount of computational resources is code optimization.***
- ***There are many reasons why we want to do this, but here are two common ones:***
  1. ***To get work done faster.***

Motivation

- ***Maximizing computational results while minimizing the amount of computational resources is code optimization.***
- ***There are many reasons why we want to do this, but here are two common ones:***
  1. ***To get work done faster.***
  2. ***To approach larger and more complex versions of the problems.***

Code Optimization

There are many ways to optimize code, but common methods include

- ***Automatic compiler optimization***
- ***Use of profilers***
- ***Algorithm analysis***
- ***Library use***

Optimization process

- ***Establish the correctness of your solution.***

Optimization process

- ***Establish the correctness of your solution.***
- ***Establish a baseline performance. Benchmarking***

Optimization process

- ***Establish the correctness of your solution.***
- ***Establish a baseline performance. Benchmarking***
- ***Find the bottleneck for performance. Profiling***

Optimization process

- ***Establish the correctness of your solution.***
- ***Establish a baseline performance. Benchmarking***
- ***Find the bottleneck for performance. Profiling***
- ***Take action to address the bottleneck. Optimization proper***

Warning

Avoid assuming what the bottleneck is!

Optimization process

- ***Establish the correctness of your solution.***
- ***Establish a baseline performance. Benchmarking***
- ***Find the bottleneck for performance. Profiling***
- ***Take action to address the bottleneck. Optimization proper***
- ***Go back to step 2, if needed.***

The $\mathbf{Ax} = \mathbf{b}$ in science

Problems that can be expressed as systems of linear equations can be found in many areas of science and computing.

- ***Chemistry***
- ***Network Analysis***
- ***Fiance***
- ***Electrical Engineering***
- ***Chemical Engineering***
- ***Mechanical Engineering***
- ***And Many more***

The problem

We want to solve large systems of linear equations; that is

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n &= b_3 \\
&\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n
\end{aligned}
$$

Matrix Representation

- ***The coefficients can be represented as a matrix***

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{nn} \end{bmatrix}, \tag{1}$$

Matrix representation

- ***the variables***

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \tag{2}$$

- ***and solutions as***

$$\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}. \tag{3}$$

Solving Linear Equations Directly

- ***We may think that we want to solve the equation in the same way as we would solve a singe linear equation*** $ax = b$***:***

$$x = a^{-1}b$$

- ***However, the calculation of matrix inverses is a costly process that involved many calculations.***
- ***In essence, it involves more operations than solving the system directly.***
- ***Even a sparse matrix*** A ***may generate a dense*** A^{-1}***.***
- ***Direct solvers are rarely used for these reasons.***

Iterative Solvers

- ***Alternatively, we can use a trial solution to generate a better approximation to the "true" solution.***

Iterative Solvers

- ***Alternatively, we can use a trial solution to generate a better approximation to the "true" solution.***
- ***That is***

$$\mathbf{A}\mathbf{x}^{trial} = \mathbf{b}^{trial}$$

Iterative Solvers

- ***Alternatively, we can use a trial solution to generate a better approximation to the "true" solution.***
- ***That is***

$$Ax^{trial} = b^{trial}$$

- ***Which can then be used to generate further refined answers until*** $|b^{\text{trial}} - b|$ ***is small enough.***

Iterative Solvers

- ***Alternatively, we can use a trial solution to generate a better approximation to the "true" solution.***
- ***That is***

$$\mathbf{A}\mathbf{x}^{trial} = \mathbf{b}^{trial}$$

- ***Which can then be used to generate further refined answers until*** $|\mathbf{b}^{\text{trial}} - \mathbf{b}|$ ***is small enough.***
- ***The method for generating new*** $\mathbf{x}$ ***vary .***

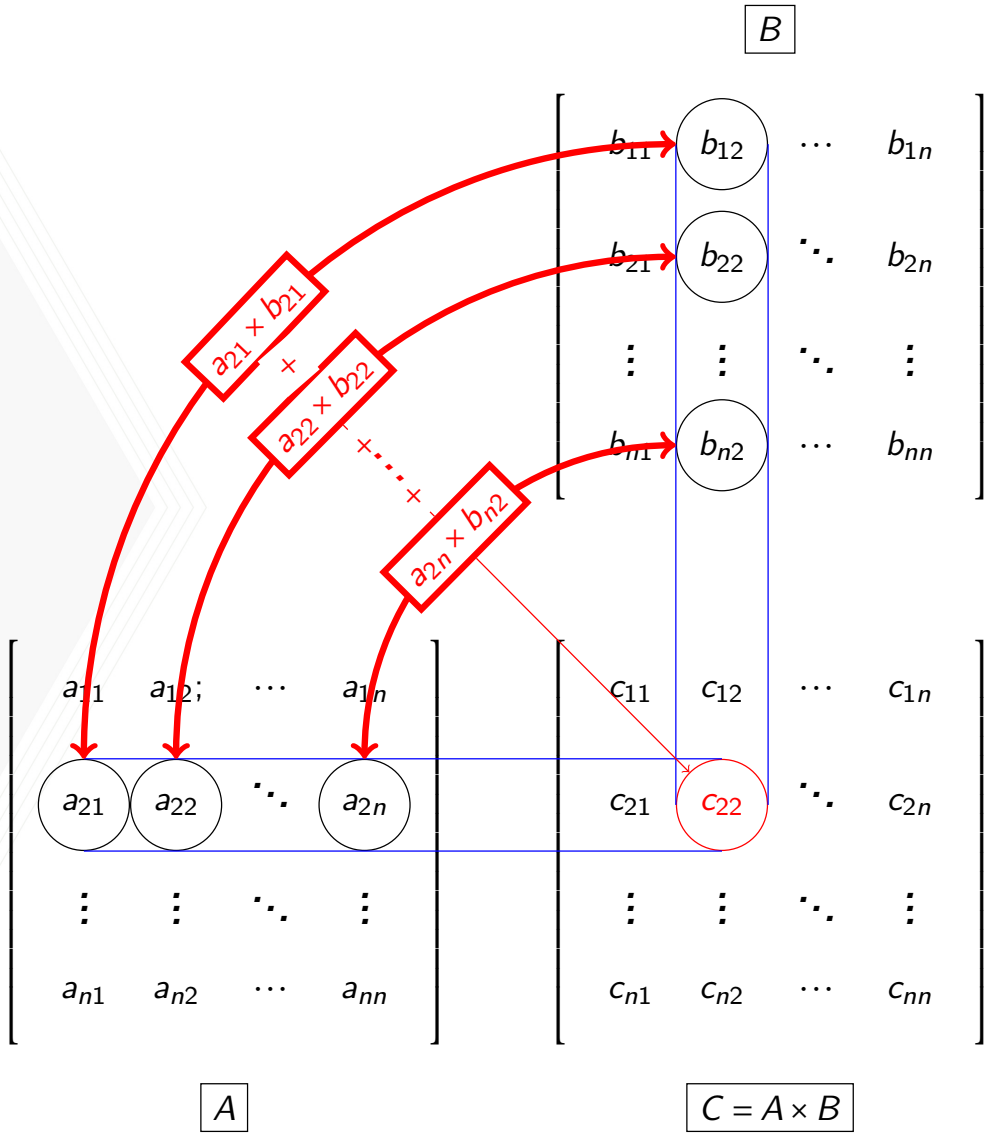
Jacobi method

1. ***Choose a convergence value,*** ϵ***. (What is the difference between solutions that will be accurate enough?) Choose a maximum number of iterations*** N
2. ***Decompose*** A ***into*** $\mathsf{L}, \mathsf{D}, \mathsf{U}$***.***
3. ***Select a guess for the solution*** x^0***.***
4. ***Calculate the next iterative of the solution as***

$$\mathsf{x}^{k+1} = -\mathsf{D}^{-1}(\mathsf{L}+\mathsf{U})\mathsf{x}^k + \mathsf{D}^{-1}\mathsf{b} \tag{4}$$

5. ***Calculate the*** $r = |\mathsf{x}^{k+1} - \mathsf{x}^k|$***.***
6. ***Continue until*** $r \leq \epsilon$ ***or the number of iterations is greater than*** N***.***

Matrix Multiplication



Some decompositions of a matrix

Decompose A

$$\mathbf{A} = (\mathbf{L} + \mathbf{U}) + \mathbf{D}, \tag{5}$$

Some decompositions of a matrix

Decompose A

$$\mathbf{A} = (\mathbf{L} + \mathbf{U}) + \mathbf{D}, \tag{5}$$

with the strictly lower matrix

$$\mathbf{L} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & 0 \end{bmatrix} \tag{6}$$

Some decompositions of a matrix

Decompose A

$$\mathbf{A} = (\mathbf{L} + \mathbf{U}) + \mathbf{D}, \tag{5}$$

with the strictly upper matrix

$$\mathbf{U} = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{6}$$

Some decompositions of a matrix

Decompose A

$$\mathbf{A} = (\mathbf{L} + \mathbf{U}) + \mathbf{D}, \tag{5}$$

with the diagonal component

$$\mathbf{D} = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, \tag{6}$$

Working on PACE Resources

- ***If you have PACE resources that you wish to work on, you should connect with X11 forwarding, like this***

  ```
  $ ssh -XY userid@login-s.pace.gatech.edu
  ```

- ***Let's start a VNC session like this***

  ```
  $ pace-vnc-job -q pace-training -l walltime=02:00:00 -l nodes=1:ppn
      =1
  ```

  Then, we'll following the off-screen prompts.

Git Repository for Presentation and Code

- ***The presentation and code are available at***

  `https://github.com/chrisblanton/gatech_optimization101`

- ***The code can be obtained by***

```
$ module load git
$ git clone https://github.com/chrisblanton/gatech_optimization101
```

Building the code

- ***Once you are in a compute session, load the GCC module like this***

```
$ module load gcc
```

- ***Then change into the desired directory and run***

```
$ make benchmark_solver.out
```

- ***To run,***

```
$ cd ./bin
$ ./benchmark_solver.out
Enter n: 1000
  Computation_time:      2.52361608
 Number_iterations:                9
 Error:      1.50885724622962 24E-007
 Number_of_ops:      19990000000.000000
 Operations per second:      7921173190.3057709
```

Let's do this!

Let's try some cases to see!

Let's do this!

Let's try some cases to see!

- ***We implement the standard matrix multiplication algorithm ourselves. I'll talk more about what that is later.***

Let's do this!

Let's try some cases to see!

- ***We implement the standard matrix multiplication algorithm ourselves. I'll talk more about what that is later.***

Let's do this!

Let's try some cases to see!

- ***We implement the standard matrix multiplication algorithm ourselves. I'll talk more about what that is later.***
- ***We want to do*** 1.0×10^6 ***in production so let's see how long 1000 takes.***

Let's do this!

Let's try some cases to see!

- ***We implement the standard matrix multiplication algorithm ourselves. I'll talk more about what that is later.***
- ***We want to do*** 1.0×10^6 ***in production so let's see how long 1000 takes.***
- ***About 4 seconds with my implementation.***

Let's do this!

Let's try some cases to see!

- ***We implement the standard matrix multiplication algorithm ourselves. I'll talk more about what that is later.***
- ***We want to do*** 1.0×10^6 ***in production so let's see how long 1000 takes.***
- ***About 4 seconds with my implementation.***
- ***Let's try 2000 then.***

Let's do this!

Let's try some cases to see!

- ***We implement the standard matrix multiplication algorithm ourselves. I'll talk more about what that is later.***
- ***We want to do*** 1.0×10^6 ***in production so let's see how long 1000 takes.***
- ***About 4 seconds with my implementation.***
- ***Let's try 2000 then.***
- ***It took about 60 seconds.***

Let's do this!

Let's try some cases to see!

- ***We implement the standard matrix multiplication algorithm ourselves. I'll talk more about what that is later.***
- ***We want to do*** 1.0×10^6 ***in production so let's see how long 1000 takes.***
- ***About 4 seconds with my implementation.***
- ***Let's try 2000 then.***
- ***It took about 60 seconds.***

What does that mean for my bigger cases

It didn't take twice as long, it took 15 times as long!

Taking action

- ***We are going to have see what we can do if we want to be able to do larger problems.***

Taking action

- ***We are going to have see what we can do if we want to be able to do larger problems.***
- ***We will first try to find out how bad the problem is (establish our baseline profile).***

Taking action

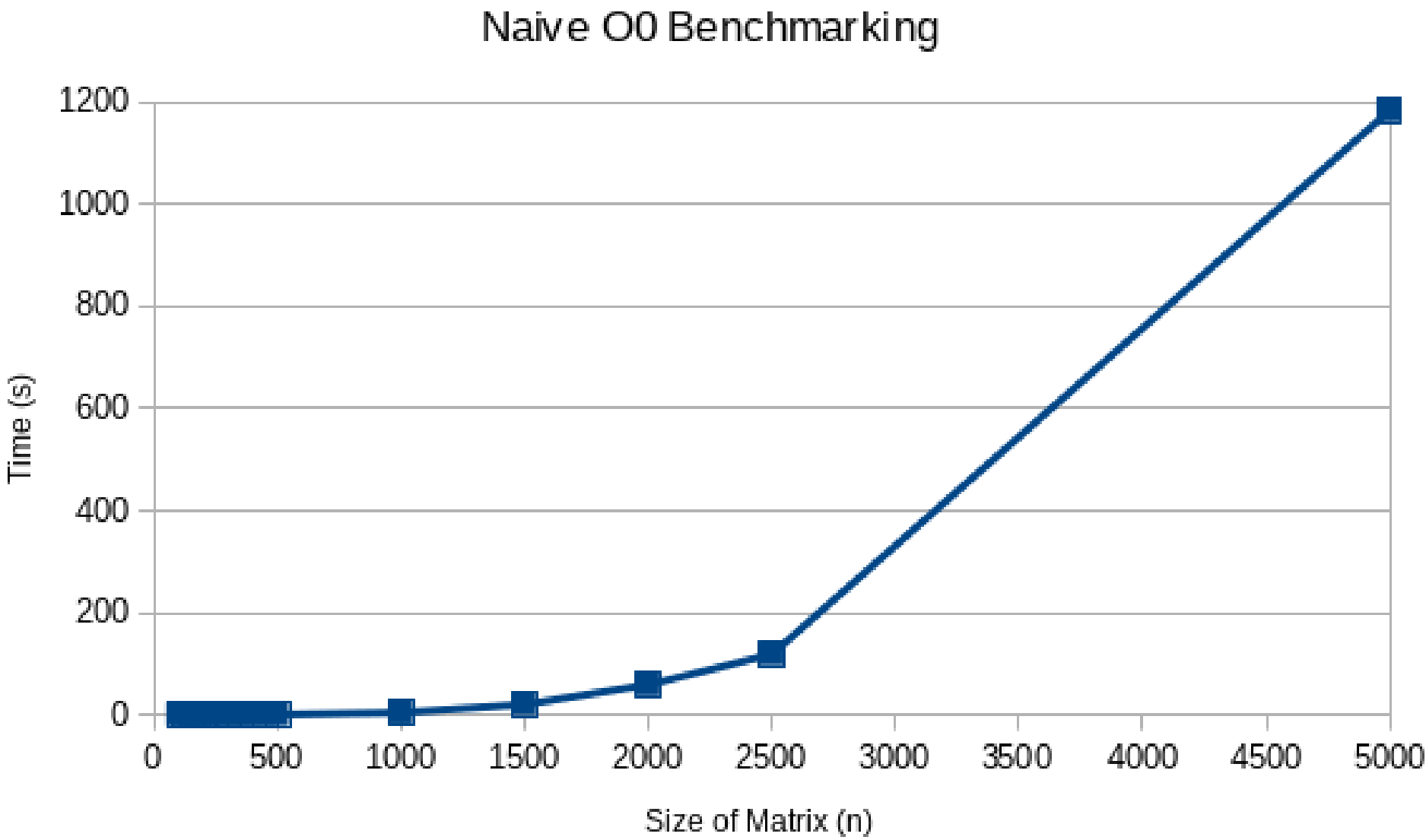
- ***We are going to have see what we can do if we want to be able to do larger problems.***
- ***We will first try to find out how bad the problem is (establish our baseline profile).***
- ***Then we'll try to find ways to improve the performance while getting the same answer.***

Benchmarking

- ***The most basic benchmarking is use the time command to determine the amount of time that is being used.***
- ***The way to run this on most Unix-like systems is to run it in this manner:***

```
$ /usr/bin/time ./benchmark_solver.out < mytemp
Enter n:
Number_iterations:                23
Error:      5.028474037160890 5E-007
Number_of_ops:      45600.000000000000
0.00 real              0.00 user              0.00 sys
```

Benchmarking results



Profiling and Optimization

Methods for optimizing

- ***Compiler optimizations.***
- ***Profiling***

Methods for optimizing

- ***Compiler optimizations.***
- ***Profiling***
- ***Simple alterations to the algorithm.***

Methods for optimizing

- ***Compiler optimizations.***
- ***Profiling***
- ***Simple alterations to the algorithm.***
- ***Using a library***

Compiler Optimizations

Compilers may, when directed, perform a number of optimizations such as

- ***Unroll loops (since loops require a logic check for each iteration).***
- ***Fuse loops.***
- ***Interchange loops.***
- ***Eliminate dead code (remove logic checks for a branch which will never be taken).***
- ***Reorder instructions to improve register reuse and more.***
- ***Strength reduction (turn instructions into simpler equivalent instructions).***

Compiler Optimizations

Compilers may, when directed, perform a number of optimizations such as

- ***Unroll loops (since loops require a logic check for each iteration).***
- ***Fuse loops.***
- ***Interchange loops.***
- ***Eliminate dead code (remove logic checks for a branch which will never be taken).***
- ***Reorder instructions to improve register reuse and more.***
- ***Strength reduction (turn instructions into simpler equivalent instructions).***

Warning

All my problems are solved! No, sometimes it does the best thing possible, other times it does not.

Compiler flags

- ***Optimization Flags***
  - ***O0*** ***No optimization, used for debugging generally***
  - ***O1*** ***Minimal optimization, some***
  - ***O2*** ***Standard optimization***
  - ***O3*** ***High optimization***
- ***Debugging Flags***

This are changed in the Makefile.

Compiler flags

- ***Optimization Flags***
  - ***O0*** ***No optimization, used for debugging generally***
  - ***O1*** ***Minimal optimization, some***
  - ***O2*** ***Standard optimization***
  - ***O3*** ***High optimization***
- ***Debugging Flags***

This are changed in the Makefile.

Effects of optimization flags on naïve MM

N	O0	O1	O2	O3
1000	4.3	2.0	2.1	2.1
1500	20.3	19.6	19.7	19.4
2000	58.8	53.5	55.4	54.2
2500	118.5	111.9	109.3	110.4
5000	1184.2	1127.0	1131.9	1132.0

Changing of optimization levels did not have a large effect on the walltime, particularly as the systems increased in size.
We need to do something ourselves!

Effects of optimization flags on naïve MM

N	O0	O1	O2	O3
1000	4.3	2.0	2.1	2.1
1500	20.3	19.6	19.7	19.4
2000	58.8	53.5	55.4	54.2
2500	118.5	111.9	109.3	110.4
5000	1184.2	1127.0	1131.9	1132.0

Changing of optimization levels did not have a large effect on the walltime, particularly as the systems increased in size.

We need to do something ourselves!

Profiling with gprof

- ***The GNU project profiler is a freely available profiling tool.***
- ***It has several modes:***

  Flat profile: ***which shows the CPU time spend in each function, number of calls for function, and finds hotspots.***

  Call Graph: ***which shows the number of times a function was called by other functions, useful to find function relations.***

  Annotated source: ***indicates the number of times a line was executed.***

Instruction on using grof

1. ***Build the program like this***

```
$ gcc -g -pg -o exeFile ./srcFile.c
$ gfortran -g -pg -o exeFile ./srcFile.f90
```

2. ***Run the executable to generate gmon.out.***
3. ***Execute gprof. The output comes to stdout, so you may wish to redirect.***

```
$ gprof ./exeFile gmon.out > profile.txt
$ gprof -l ./exeFile gmon.out > profile_line.txt
$ gprof -A ./exeFile gmon.out > profile_anotated.txt
```

Profile of our MM with gprof

```
Flat profile:

Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  s/call   s/call  name
99.38      4.27     4.27       12     0.36     0.36  __mod_matops_MOD_dgemm
0.23       4.28     0.01        1     0.01     0.01  __mod_matops_MOD_generatediagdommat
0.23       4.29     0.01        1     0.01     0.01  __mod_matops_MOD_strictlowerpart
0.23       4.30     0.01        1     0.01     0.01  __mod_matops_MOD_strictupperpart
0.00       4.30     0.00  1001082     0.00     0.00  __mod_random_MOD_generate_random_a_b
0.00       4.30     0.00        1     0.00     4.30  MAIN__
#Snip
```

The main function that takes most of the time is the dgemm (which is the matrix multiplication).

Profiling with Intel Vtune Amplifier

- ***Intel Vtune Amplifier is a commercial component of the Intel Parallel Studio suite.***
- ***Amplifier offers a GUI for setting up runs and for analyzing results.***
- ***There are many options and good documentation***
  `https://software.intel.com/en-us/vtune-amplifier-help`

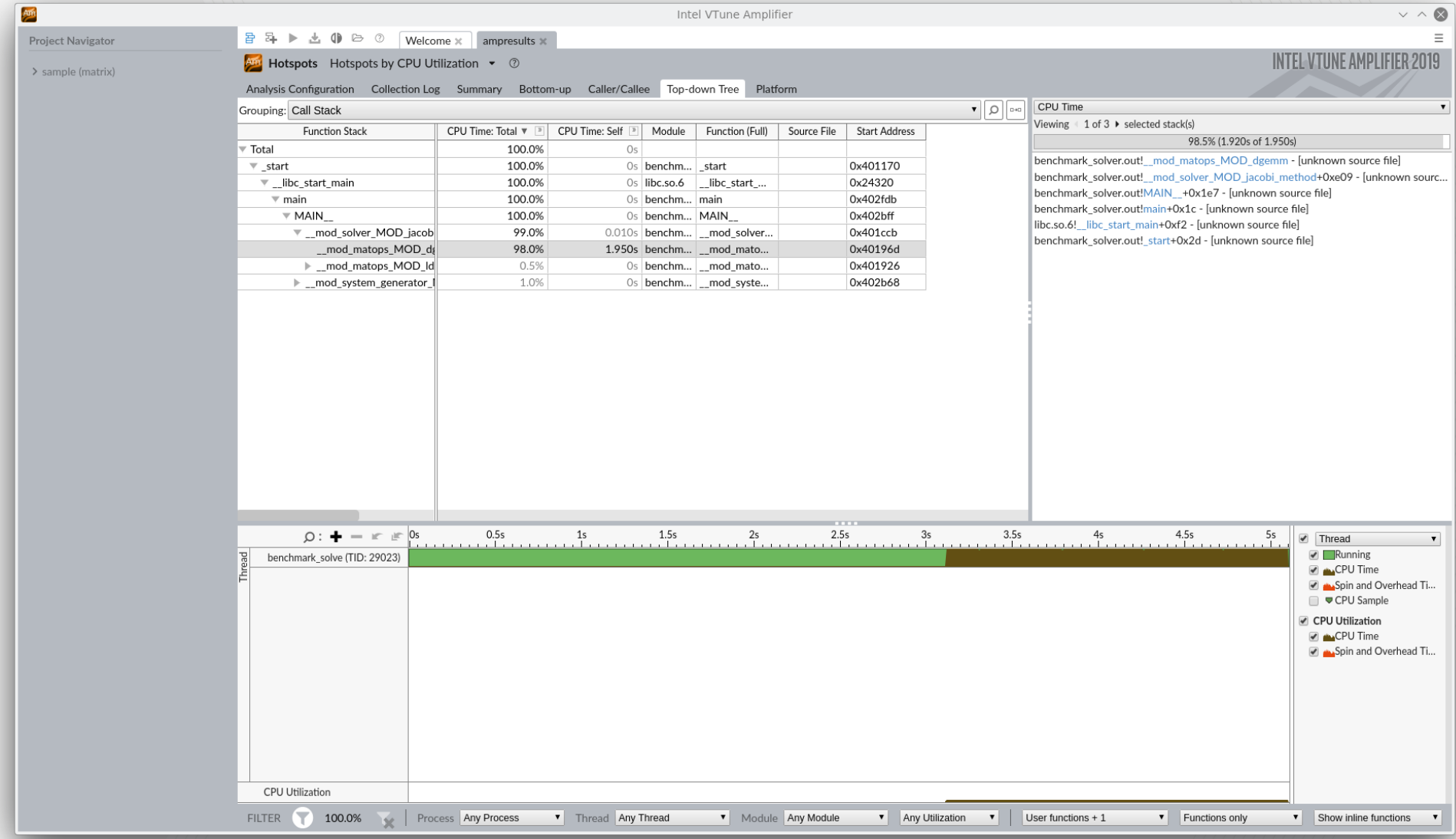
Using Amplifier on the command-line

- ***We will do a hotspot analysis of the naïve code using Amplifier.***
- ***With the amplxe-cl in our path***

```
$ module purge
$ module load intel/19.0
$ amplxe-cl -collect hotspots -result-dir ampresults ./
    benchmark_solver.out
```

- ***The program will run and results will be collected in the ampresults (in this case) directory.***
- ***It is often to then analyze results using the GUI.***

Profile of our MM with Amplifier



The Issue

- ***We have seen that the issue appears to be the matrix multiplication function within the code.***

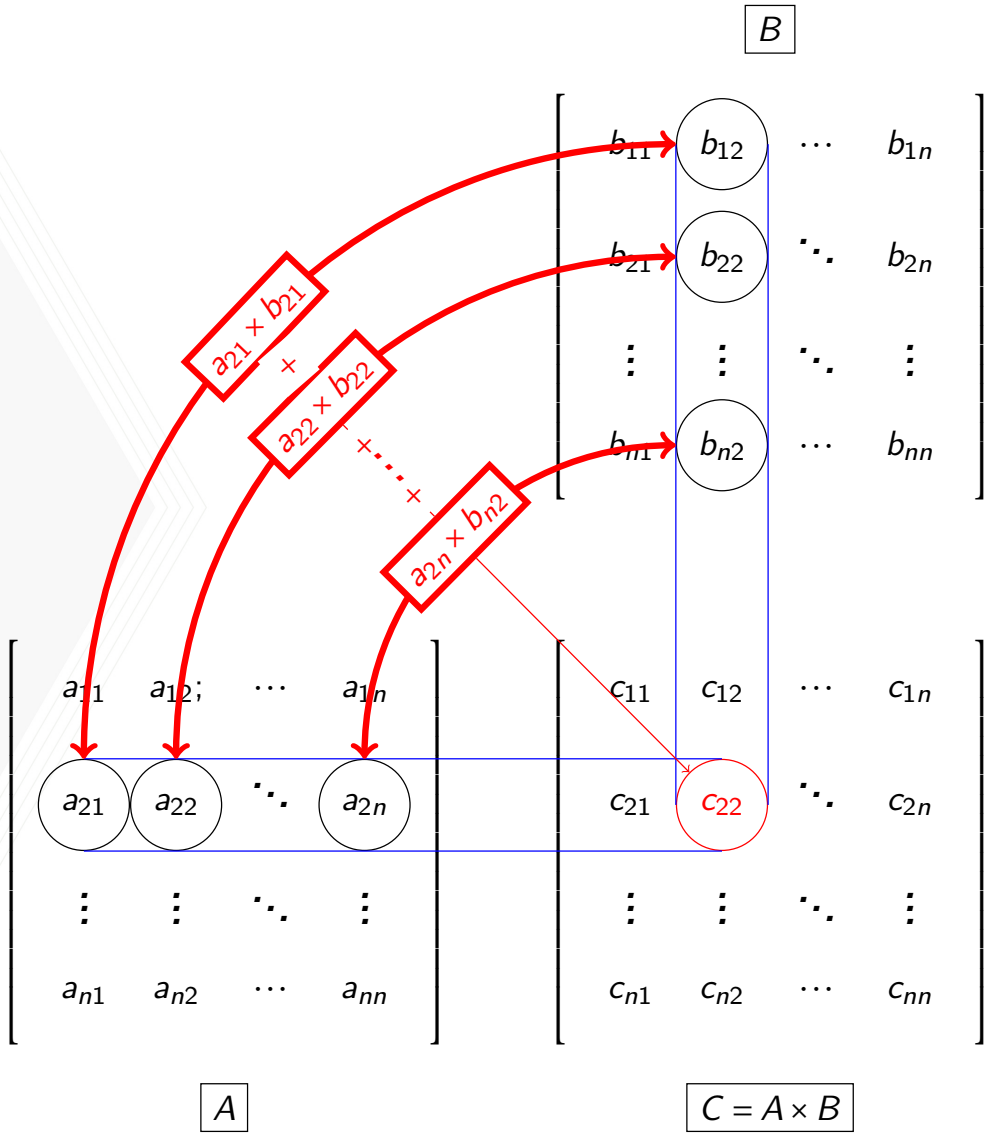
The Issue

- ***We have seen that the issue appears to be the matrix multiplication function within the code.***
- ***Automatic solutions are not enough to solve this issue.***

The Issue

- ***We have seen that the issue appears to be the matrix multiplication function within the code.***
- ***Automatic solutions are not enough to solve this issue.***
- ***We will have to change the function to improve the performance.***

Matrix Multiplication



How many operations in a Matrix Multiplication

For a pair of $n \times n$ ***matrices, we must perform***

Multiplication Operations n^3

Addition Operations $n^3 - n^2$

Total number of Operations $2n^3 - n^2$

Considering the Algorithm

The math that is implemented in the matrix multiplication is

$$C_{ij} = \alpha \times \sum_{k=1}^{n} A_{ik} B_{kj} + \beta \times C_{ij} \tag{7}$$

This is a little different, but it was used align with BLAS (more on that later!)

Considering the Algorithm

```
do i = 1, n
        do j = 1, n
                do k = 1, n
                        C(i,j) = alpha*A(i,k)*B(k,j) + beta*C(i,j)
                end do
        end do
end do
```

Considering the Algorithm

```
do i = 1, n
        do j = 1, n
                do k = 1, n
                        C(i,j) = alpha*A(i,k)*B(k,j) + beta*C(i,j)
                end do
        end do
end do
```

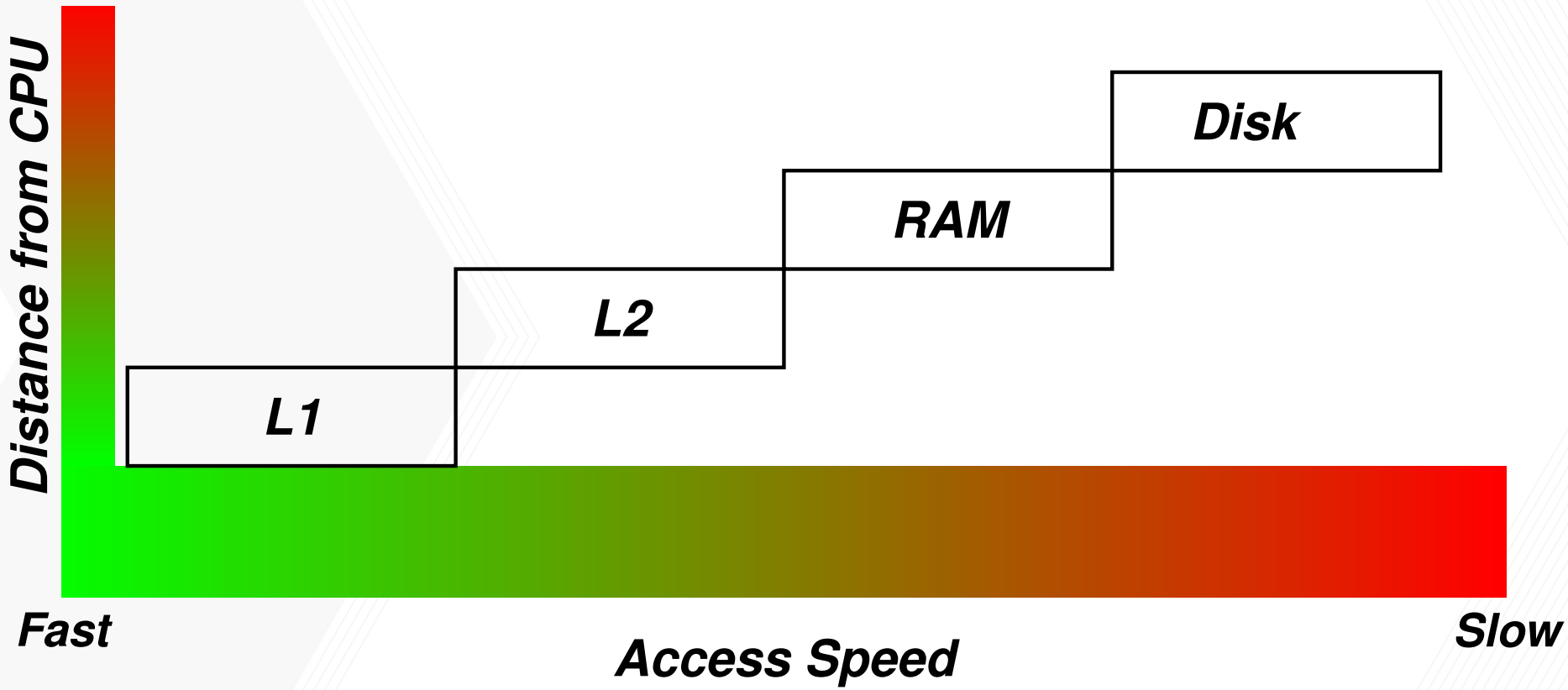
- ***This algorithm does not take into account how the values are stored in memory.***

Considering the Algorithm

```
do i = 1, n
        do j = 1, n
                do k = 1, n
                        C(i,j) = alpha*A(i,k)*B(k,j) + beta*C(i,j)
                end do
        end do
end do
```

- ***This algorithm does not take into account how the values are stored in memory.***
- ***Analysis of the algorithm shows that it scales as*** $O(n^3)$

Computer memory speeds



Slow memory access in the naïve MM

Analysis of a simple model that assumes two speeds of memory-fast and slow.

$$
\begin{aligned}
m &= n^3 && \textit{to read each column of } \mathrm{B}\, n \textit{ times} \\
&+ n^2 && \textit{to reach each row of } \mathrm{A} \textit{ once} \\
&+ 2n^2 && \textit{to read and write each element of } \mathrm{C} \textit{ once} \\
&= n^3 + 3n^2
\end{aligned}
$$

Slow memory access in the naïve MM

Analysis of a simple model that assumes two speeds of memory-fast and slow.

$$
\begin{aligned}
m &= n^3 && \textit{to read each column of } \mathrm{B}\, n \textit{ times} \\
&+ n^2 && \textit{to reach each row of } \mathrm{A} \textit{ once} \\
&+ 2n^2 && \textit{to read and write each element of } \mathrm{C} \textit{ once} \\
&= n^3 + 3n^2
\end{aligned}
$$

This is a bottleneck to address.

Considerations

- ***There are multiple layers of memory, called cache, with varying degrees of speed.***
- ***When a variable is not in fastest memory, this is called a cache-miss.***
- ***Cache-misses require multiple instructions to free space and load the needed variable.***
- ***This can proceed for several cache levels.***

Tiling of the MM algorithm

- ***In order to take the cache behavior into account, we can use tiling.***

Tiling of the MM algorithm

- ***In order to take the cache behavior into account, we can use tiling.***
- ***This is an implementation of a block matrix multiplication***

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

Tiling of the MM algorithm

- ***In order to take the cache behavior into account, we can use tiling.***
- ***This is an implementation of a block matrix multiplication***

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

- ***We do this by converting the three-loop implementation into a six-loop implementation with loops to handle the tiling.***

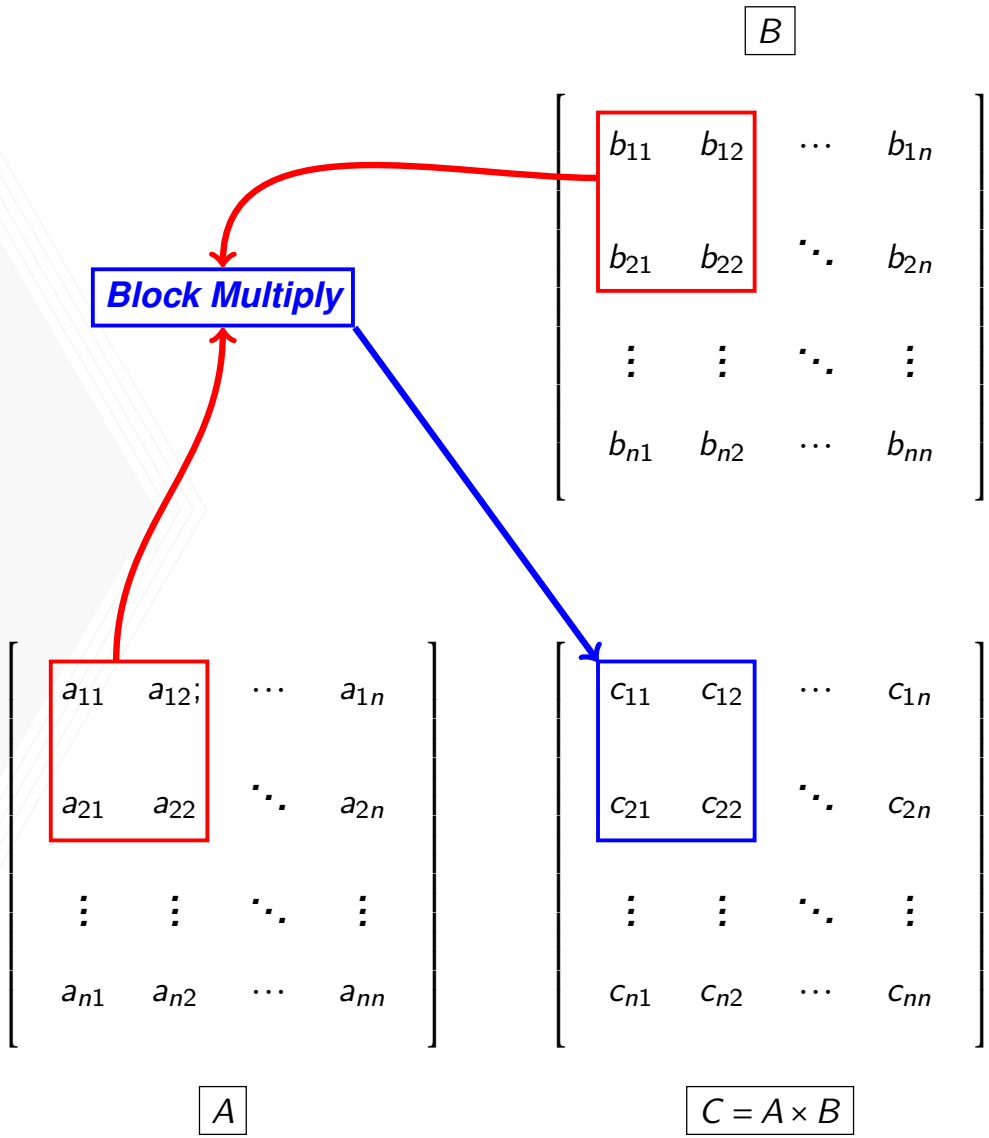
Tiling of the MM algorithm

- ***In order to take the cache behavior into account, we can use tiling.***
- ***This is an implementation of a block matrix multiplication***

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

- ***We do this by converting the three-loop implementation into a six-loop implementation with loops to handle the tiling.***
- ***This still has the same number of operations as the naïve implementation.***

Graphical tiling of MM



Tiled version of MM

```
do i0 = 1, n, tile_size
do j0 = 1, m, tile_size
do k0 = 1, k, tile_size
 do i1 = i0, min(i0+tile_size-1,m)
 do j1 = j0, min(j0+tile_size-1,n)
 do k1 = k0, min(k0+tile_size-1,k)
  C(i1,j1) = alpha*(A(i1,k1)*B(k1,j1))+beta*C(i1,j1)
 end do
 end do
 end do
end do
end do
end do
```

Memory access in the tiled version of MM

The memory traffic m between slow and fast memory is

$$
\begin{aligned}
m &= Nn^2 \\
&+ Nn^2 \\
&+ 2n^2 \\
&= (2N+2)n^2
\end{aligned}
$$

read each block of B N^3 ***times***

read each block of A N^3 *times*

read and write each block of C *once*

$(N^2 b^2 = N^3 (n/N)^2 = Nn^2)$

Building and running the tiled version of the solver

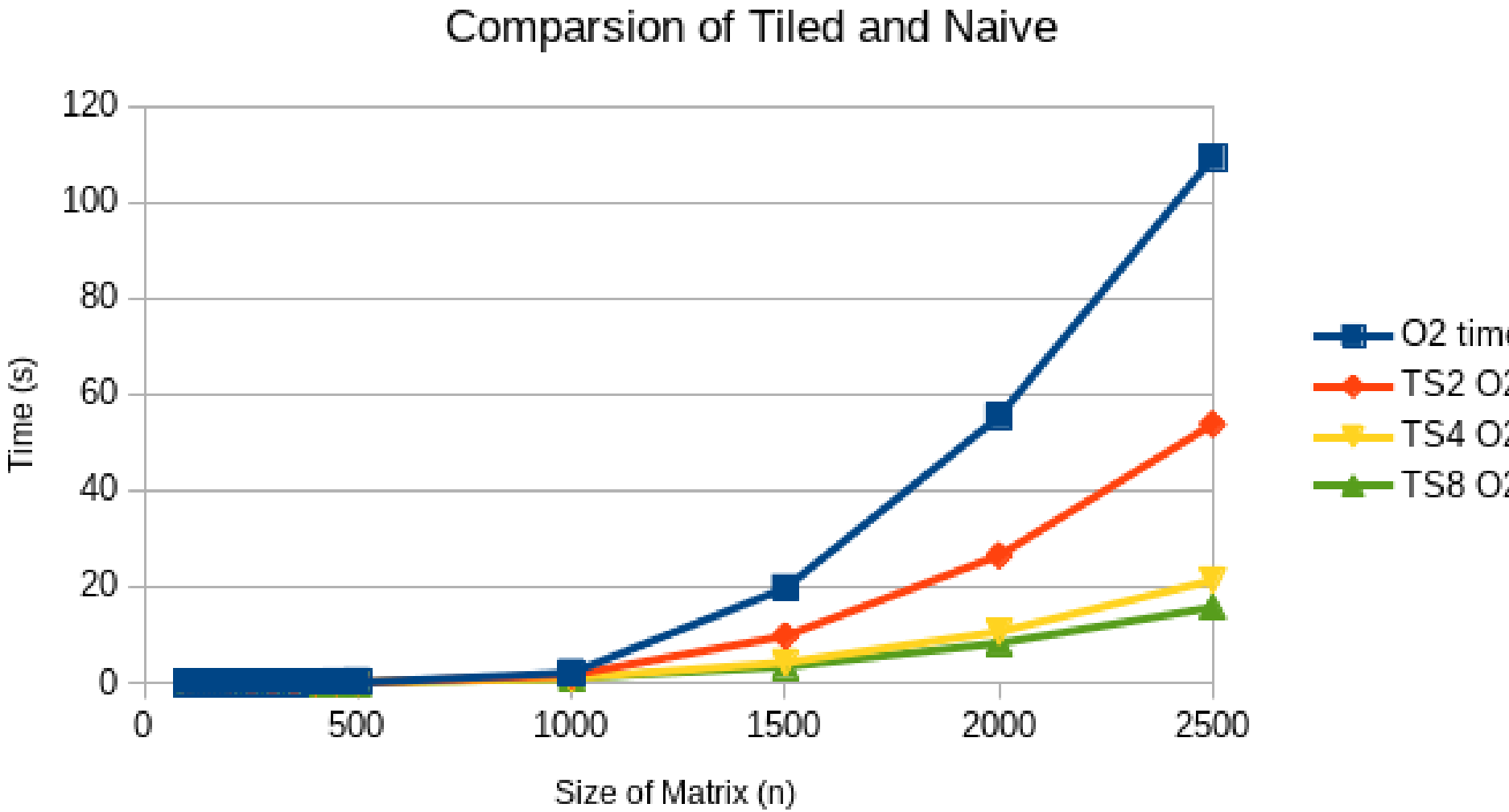
- **To build the optimized code**

```
$ cd ${GITROOT}/opt1
$ module purge; module load gcc
$ make benchmark_solver_tilesize.out
```

- **To run**

```
$ cd ./bin
$ ./benchmark_solver_tilesize.out
Enter n:
1000
Enter tile_size:
4
Computation_time:       1.90570998
Number_iterations:                5
Error:      2.30597052564007 75E-009
Number_of_ops:       11994000000.000000
```

Results of using tiled MM



Using a library

- ***When possible, using a library is very good idea.***
- ***Matrix operations have been optimized in the BLAS (Basic Linear Algebra System) library.***
- ***There are several implementations of the standard, but many give much better performance that a naïve or optimization outside of specialist knowledge.***

How to use BLAS with GCC on PACE

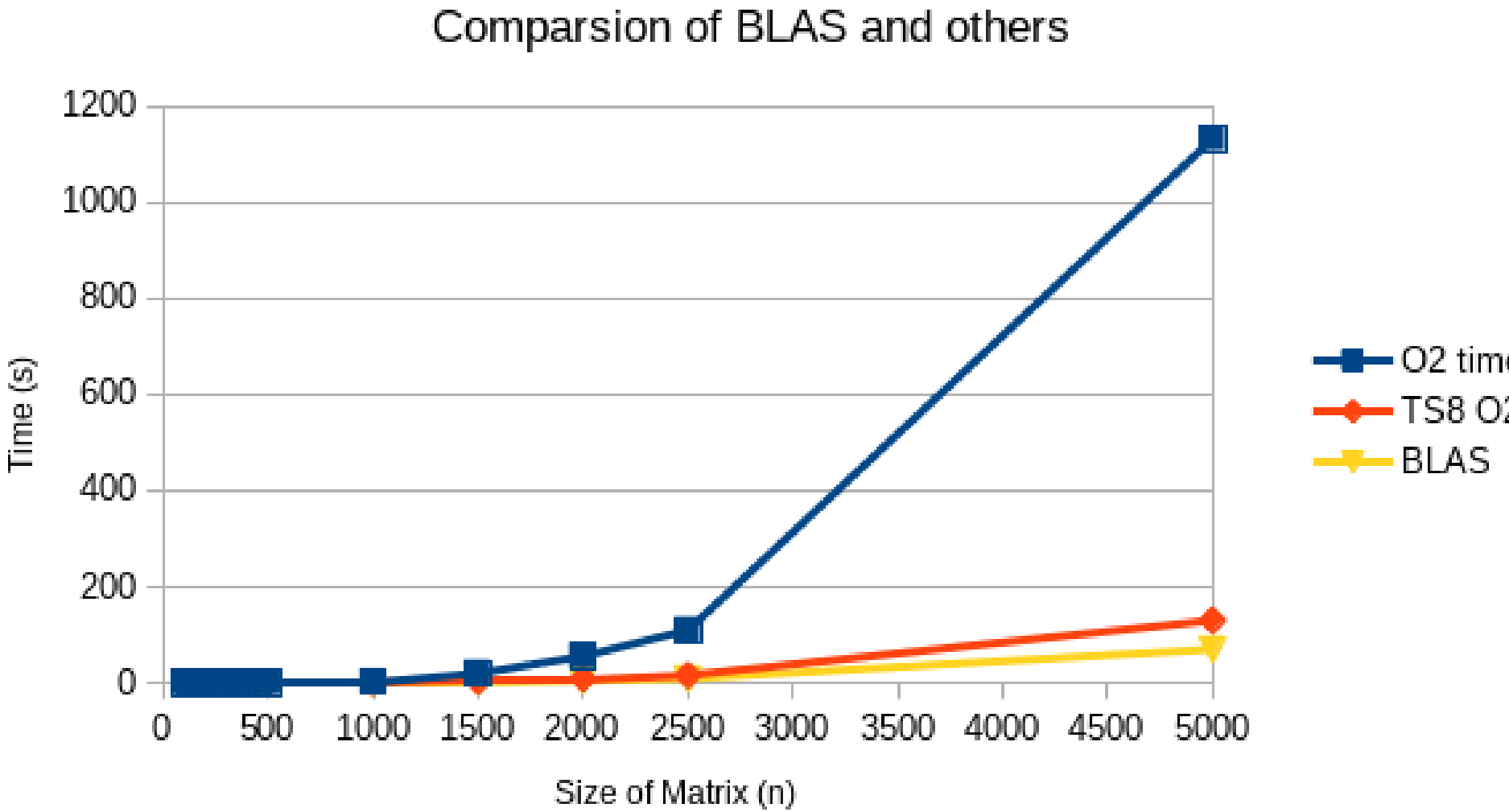
- **Load the environment using the module system**

```
$ cd ${GITROOT}/blas
$ module purge; module load gcc
$ make benchmark_solver.out
```

- **To run the application,**

```
$ cd ./bin
$ ./benchmark_solver.out
Enter n:
1000
Number_iterations:                    9
Error:      1.5088572462296224E-007
Number_of_ops:      19990000000.000000
Total_executation_time:     0.666899025
Operations per second:      29974552724.536854
```

Results of using library



Conclusion

Conclusions

- ***In this workshop, some basics the following topics where covered.***
  - ***Benchmarking***
  - ***Profiling***
  - ***Code Optimization***
  - ***Library use***

Future training

- ***Shared-memory parallelization***
- ***Distributed-memory parallelization***
- ***GPU programming***

Please fill out the survey

Please fill out the survey

https://b.gatech.edu/2lke9g8