

Assignment 1

Exercise 1: Compiling and running on Dardel.

Task 1.1

Describe all steps to connect to Dardel, compile the code, and execute the code on computing nodes with SLURM (Interactive and batch).

- PDC Account
- Kerberos and ssh
- Compiling
- Running on interactive node
- Running on batch job

To connect to dardel, use the following steps: - Apply a PDC account, after which you will be sent a password for your account - PDC uses the kerberos authentication protocol and uses ssh to connect to dardel.

1. Create a kerberos ticket using ``kinit --forwardable user@NADA.KTH.SE``
2. ssh into Dardel using this ticket with ``ssh YourUsername@dardel.pdc.kth.se``

- Compiling

1. Code can be compiled using the `cc`, `CC` wrappers for `gcc`.

- Running on an interactive node

```
$ cd /cfs/klemming/nobackup/u/user
$ salloc --nodes=1 -t 01:00:00 -A edu23.DD2356 -p main
$ srun -n 128 ./hello.out
```

- Running on a batch job

We create a file `job.sh`:

```
#!/bin/bash -l
# The -l above is required to get the full environment with modules

# The name of the script is myjob
#SBATCH -J myjob
# Only 1 hour wall-clock time will be given to this job
#SBATCH -t 0:01:00
#SBATCH -A edu23.DD2356
# Number of nodes
#SBATCH -p main
#SBATCH --nodes=1
#SBATCH -e error_file.e

# Run the executable file
# and write the output into my_output_file
srun -n 128 ./hello.out > hello_output
```

Then we submit it with

```
sbatch job.sh
```

For both, the output looks something like

```
Hello world from processor nid001264, rank 86 out of 128 processors
Hello world from processor nid001264, rank 16 out of 128 processors
Hello world from processor nid001264, rank 97 out of 128 processors
...
...
```

...

Hello world from processor nid001264, rank 99 out of 128 processors

Hello world from processor nid001264, rank 127 out of 128 processors

Hello world from processor nid001264, rank 115 out of 128 processors

Task 1.2

How many computing nodes does Dardel have?

- Dardel supercomputer have 1270 nodes. [736 NAISS thin, 268 NAISS large, 8 NAISS Huge, 10 NAISS Giant, 36 256GB KTH nodes, 248 512GB KTH nodes]

How many CPUs, CPU core, and memory does each computing node have?

- 2 CPUs, 128 Physical cores with 256 virtual cores. Memory size varies with the type of node. There are several memory sizes in the CPU partition:
 - (1) 736×256 GB (NAISS thin nodes)
 - (2) 268×512 GB (NAISS large nodes)
 - (3) 8×1024 GB (NAISS huge nodes)
 - (4) 10×2048 GB (NAISS giant nodes)
 - (5) 36×256 GB (KTH industry/business research nodes)
 - (6) 248×512 GB (KTH industry/business research nodes)

What is the total number of cores and memory of the Dardel computer?

- 162,560 Total cores in the CPU partition. With 157,928 GB of memory.

What is the fastest supercomputer in the world? What are the fastest European and Asian supercomputers? What is their power usage?

- Frontier in the world [21,100.00 kW]. Asian - Supercomputer Fugaku [29,899kW]. European - LUMI [6,016kW].

Exercise 2: Sustainability and supercomputers

Task 2.1

Using the calculator at <http://calculator.green-algorithms.org/>. Evaluate the power usage and carbon footprint of running a simulation on 10 Dardel computing nodes, assuming that we only use CPU, all the memory on the nodes, and we neglect the network costs. The simulation runs on 10 computing nodes for 12 hours.

Thermal Design Power (TDP) value of AMD EPYC Zen2 is about 225 W. - 12 Hours, CPU, 128 Cores, 256gb
* 10 = 7.19kg CO2, 1.27 MWh

Exercise 3: Modeling Sparse Matrix-Vector Multiply.

Note: Following was modelled using local machine not dardel.

Task 3.1

What is the performance in total execution time - do not consider data movement - according to your performance model on Dardel or your local computer for different sparse matrices $nrows = 10^2, 10^4, 10^6$, and 10^8 ?

Laptop - i7-10750H (Base 2.6Ghz - 5.0GHz)

Taking the base rate gives $(1/2.6) \cdot 10^{-9}$ s/operations

for each nnz:

$$nnz = 460 \implies 0.0000001769s$$

$$nnz = 49600 \implies 0.00001907s$$

$$nnz = 4996000 \implies 0.0019215s$$

$$nnz = 499960000 \implies 0.192292s$$

Task 3.2

What is the measured performance in total execution time and floating-point operations per second running `spmv.c` for different sizes $= 10^2, 10^4, 10^6$, and 10^8 ? Compare the results from the performance model and experimental results. Discuss the comparison in the report.

Time for Sparse Ax, $nrows=100$, $nnz=460$, $T = 0.000002s$, $FLOPS = 2 \cdot nnz/T = 460000000$

Time for Sparse Ax, $nrows=10000$, $nnz=49600$, $T = 0.000262s$, $FLOPS = 2 \cdot nnz/T = 378625954$

Time for Sparse Ax, $nrows=1000000$, $nnz=4996000$, $T = 0.028904s$, $FLOPS = 2 \cdot nnz/T = 345696097$

Time for Sparse Ax, $nrows=100000000$, $nnz=499960000$, $T = 0.598384s$, $FLOPS = 2 \cdot nnz/T = 1671033984$

The theoretical performance model consistently underestimates the execution time, which is reasonable as the model only takes into account computing time.

Task 3.3

What is the main reason for the observed difference between the modeled value and the measured value?

The model only takes into account computing time, and not read and write times.

Task 3.4

What are the read bandwidth values you measure running `spmv.c` for different sizes $nrows = 10^2, 10^4, 10^6$, and 10^8 ?

`sizeof(int) + sizeof(double) = 12 bytes`

`Read bandwidth = (nnz + nrows)(sizeof(int) + sizeof(double)) / T = 12 * (nnz + nrows) / T`

$nrows=100$, $nnz=460$, Read bandwidth = 3360 MB/s

$nrows=10000$, $nnz=49600$, Read bandwidth = 2729 MB/s

$nrows=1000000$, $nnz=4996000$, Read bandwidth = 2489 MB/s

$nrows=100000000$, $nnz=499960000$, Read bandwidth = 12031 MB/s

Task 3.5

What is the bandwidth you obtain by running the STREAM benchmark on your system? How does it compare to the bandwidth you measured in SpMV? Discuss the comparison.

Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	28199.4	0.005750	0.005674	0.005853
Scale:	17540.2	0.009188	0.009122	0.009401
Add:	20904.5	0.011644	0.011481	0.011998
Triad:	20714.7	0.011821	0.011586	0.012412

The STREAM benchmark shows a lot higher rates for all the operations compared to our read bandwidth measured on the SpMV benchmark. However, our calculated read bandwidth does not take into account computation time.

Exercise 4: The memory mountain

Note: Following was modelled using local machine not dardel.

Task 4.1

Report the name of the processor and the size of the L1, L2, and L3 of the processor you are benchmarking. You can check the specs of your processor online.

Processor: i7-10750H

L1 = 384kB

L2 = 1.5MB

L3 = 12MB

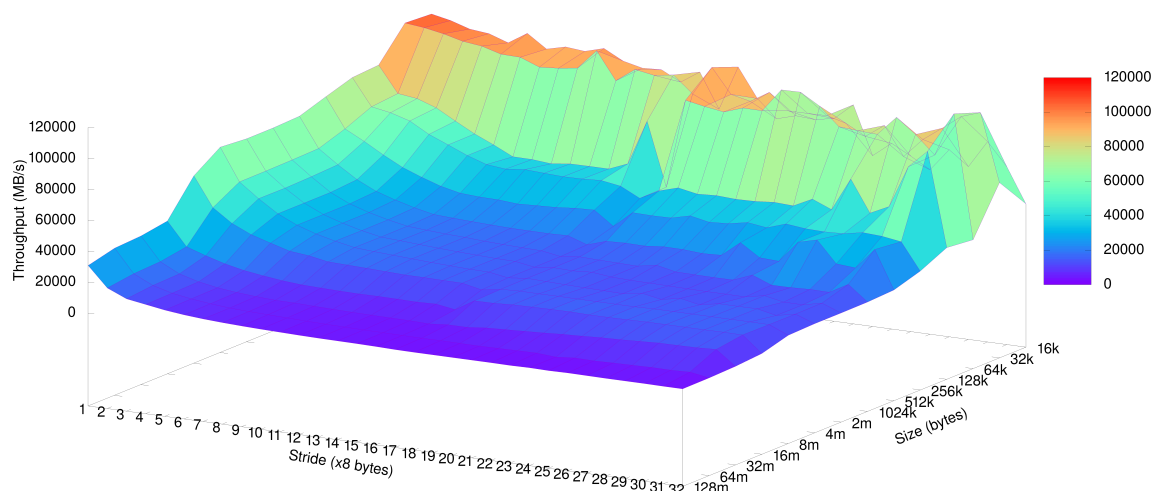


Figure 1: Memory Mountain

Task 4.2

Task 4.3

What region (array size, stride) gets the most consistently high performance (ignoring spikes in the graph that are noisy results...)? What is the read bandwidth reported?

Small arrays with low stride is the consistently most high performing region. The read bandwidth is up to 103469 MB/s.

Task 4.4

What region (array size, stride) gets the most consistently low performance (Ignoring spikes in the graph that are noisy results...)? What is the read bandwidth reported?

High stride and large array size is the consistently most low performing region. The read bandwidth is 2907 MB/s at the lowest.

Task 4.5

When you look at the graph for stride=1, you (should) see relatively high performance

compared to stride=32. This is true even for large array sizes that are much larger than the L3 cache size. How is this possible, when the array cannot possibly all fit into the cache? Your explanation should include a brief overview of hardware prefetching as it applies to caches.

Hardware prefetching loads data that is spatially local into cache memory. In this case, this means that consecutive elements of the array are loaded into cache. With a smaller stride, we can access more elements that are in the cache before there is a cache miss and prefetching has to be done again. For instance, if a cache line is 128 bytes, and we're accessing 8 byte longs we can read 16 values before a cache miss occurs with a stride of 1, but only 4 values if we have a stride of 4. The large amount of cache misses leads to the low performance when the stride length is large.

Task 4.6

What is temporal locality? What is spatial locality?

Temporally locality is the reuse of recently accessed data. Spatially locality is using data that is nearby in memory.

Task 4.7

Adjusting the total array size impacts temporal locality, why? Will an increased array size increase or decrease temporal locality?

Adjusting the array size impacts temporal locality as a smaller fraction of data is temporally local. Increasing the array size decreases temporal locality.

Task 4.8

Adjusting the read stride impacts spatial locality, why? Will an increased read stride increase or decrease spatial locality?

Read stride impacts spatial locality as memory further away that is not spatially local is read.

Exercise 5: Write a benchmark to measure performance

Task 5.1.1

[On local machine]

What is the average runtime?

The average running time is 0.00000405s

Task 5.1.2

Increase N and compile the code, what is the average running time now?

Increasing the N to 500,000 does not alter the running time of 0.00000405s

Task 5.2.1

Why is the execution time like that in the previous question when the flag -O2 is used?

Answer this question using the information you find in the assembly code.

Inspecting the ASM code shows the compiler optimized the loop out of the code.

As the compiler viewed the code as 'dead' code as it is not being used.

Task 5.2.2

What is the average execution time without the -O2 flag?

For N = 5000, the execution time with no -O2 flag is 0.00004816 s

For N = 50,000, the execution time with no -O2 flag is 0.00047493 s

Task 5.3

What is the clock granularity on Dardel or your local computer?

Granularity:

$$9.54 \cdot 10^{-7}s$$

Task 5.4.1 Modify the program that you used in question 5.1 and do the following such that the code runs properly with -O2 optimization:

The code runs as expected, inspecting the asm code shows the program executes the loop as intended. In order to make the code run as expected with -O2, we used the volatile keyword for the C array.

Task 5.4.2

Below are the average and minimum timing for the loop over 20 iterations.

- Average Execution time: 0.00006533 s
- Min Execution time: 0.00006294 s

Exercise 6: Measure the Performance of Matrix-Matrix Multiply and Transpose with perf

Task 6.1

We get the following results from perf:

	MSIZE=64 Naive	MSIZE=64 Optimised	MSIZE=1000 Naive	MSIZE=1000 Optimised
Elapsed time (seconds)	0.006008407	0.026428618	9.820340947	4.068640160
Instructions per cycle	3.05223157746	3.39699755395	2.33566861357	5.65940389993
L1 cache miss ratio	0.156965536823	0.0152186701427	0.568518889684	0.0428330419037
L1 cache miss rate PTI	45.6882622051	6.56849478495	162.654417351	18.374515994

Importantly, the average time per matrix multiplication is the following:

	MSIZE=64 Naive	MSIZE=64 Opt	MSIZE=10 Naive	MSIZE=1000 Opt
Average time (s)	0.000181	0.000154	0.889926	0.365820

What are the factors that impact the most the performance of the matrix multiply operation for different matrix sizes and implementations (naive vs optimized)?

Cache misses are the biggest performance factor between the naive and optimised implementation. The naive algorithm steps across columns of the matrix B, causing many cache misses especially when the matrix is large. The program takes a stride of `MSIZE * sizeof(double)`, which we saw was bad for performance in the memory mountain example. Rearranging the loop leads to the optimised versioning utilising spatial locality by accessing data along the rows, as the matrices are stored in row major order in memory, taking only a stride of `sizeof(double)`.

Task 6.2

Using the perf tool we measured the following performance:

	N=64	N=128	N=2048
Elapsed time (seconds)	0.015456084	0.012847589	9.393376821
Bandwidth/Rate (MB/s)	1.43e+04	6.90e+03	3.60e+02
Instructions per cycle	2.19010949691	1.62430143786	0.0975394373964
L1 cache miss ratio	0.183398257284	0.327938976196	0.296058060658
L1 cache miss rate PTI	66.5605710444	147.558995856	477.708493074

With the reported base rate for each N being:

	N=64	N=128	N=2048
Base rate (seconds)	2.29e-06	1.90e-05	9.32e-02

What are the factors that impact the performance of the transpose operation most for different matrix sizes and implementations?

Cache size impact the performance of the transpose operation most. With the size of matrixes increasing, the miss rate of cache increases and because the load and store from memory cause great execution time than from cache, the performance will be worse.

Which code transformations can be used in the code for the matrix transpose to improve the re-usage of cache?

Loop transformations can be used in the code for the matrix transpose. We can reorder the operations so that spatial locality is preserved. There are two methods to break loops into blocks: one is strip mining(Break a loop up into blocks of consecutive elements); Another is loop reordering(Move the loop over i inside the jj loop).