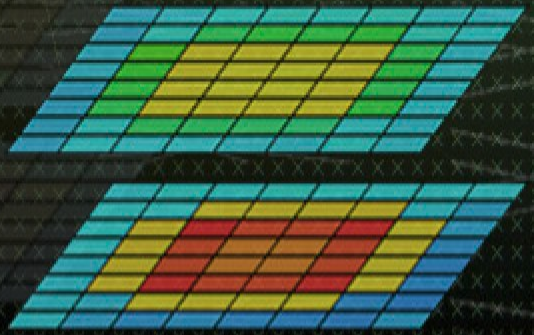


MAQAO
Hands-on exercises

Profiling bt-mz (incl. scalability)
Optimising a code

Setup

Login to the cluster (using login8 for access to /dine/data)

```
> ssh <username>@glogin-p3.hpc.gwdg.de
```

Create scratch directory and copy handson material there

```
> export SCRATCH=/scratch/lustre-grete/projects/tr_20260831-vi-hps-tuning/$USER
> mkdir $SCRATCH
> cd $SCRATCH
> tar xvf $PROJECT/training-materials/maqao/MAQAO_HANDSON.tgz
```

Load MAQAO environment

```
> module use $PROJECT/modules
> module load maqao/2026.1.0
```

Recompile bt-mz with debug symbols (if not already done)

Extract NPB and go to the extracted directory

```
> export NPB=/path/to/NPB3.4-MZ-MPI
> cd $NPB
```

Load Intel compiler and environment

```
> module load gcc/14.2.0 openmpi/4.1.7
```

Compile and run

```
> cp $SCRATCH/MAQAO_HANDSON/bt/make.def config/
> make bt-mz CLASS=C
> cd bin
> sbatch $SCRATCH/MAQAO_HANDSON/bt/reference.sbatch
```

Profiling bt-mz with MAQAO

Cédric Valensi

Launch MAQAO ONE View on bt-mz (one liner)

Launch ONE View (one liner)

```
> cd $NPB/bin
> maqao oneview -R1 envv_OMP_NUM_THREADS=48 mpi-command="srun --
reservation=vi-hps-tuning -p standard96s -N 2 --tasks-per-node=2 -c
<OMP_NUM_THREADS>" -xp=OV_btmz --show-program-output -- ./bt-mz.C.x
```

Display MAQAO ONE View results in text mode (optional)

A sample result directory is in **`MAQAO_HANDSON/bt/offline.tgz`**

Results can also be viewed directly on the console in text mode:

```
> maqao oneview -R1 -xp=OV_btmz --output-format=text
```

sshfs & scp hints

- To install sshfs on Debian-based Linux distributions (like Ubuntu)

```
> sudo apt install sshfs
```

- Recommended to close a sshfs directory after use

```
> fusermount -u /path/to/sshfs/directory
```

- scp is slow to copy directories (especially when containing many small files), copy a .tgz archive of the directory

Display MAQAO ONE View results

The HTML files are located in **<exp-dir>/RESULTS/<binary>_one_html**, where <*exp-dir*> is the path of he experiment directory (set with -xp) and <*binary*> the name of the executable.
Mount $SCRATCH locally:

```
> mkdir gwdg_NPB
> sshfs <username>@glogin-p3.hpc.gwdg.de:/path/to/NPB gwdg_NPB
> firefox gwdg_NPB/bin/OV_btmz/RESULTS/bt-mz.C.x_one_html/index.html
```

It is also possible to compress and download the results to display them:

```
> tar czf $SCRATCH/bt_html.tgz OV_btmz/RESULTS/bt-mz_C.x_one_html
```

```
> scp
<username>@glogin-p3.hpc.gwdg.de:/scratch/lustre-grete/projects/tr_20260831-
vi-hps-tuning/<username>/bt_html.tgz .
> tar xf bt_html.tgz
> firefox OV_btmz/RESULTS/bt-mz.C.x_one_html/index.html
```

(optional) Setup ONE View (with config file)

Retrieve the configuration file prepared for bt-mz in interactive mode from the MAQAO_HANDSON directory

```
> cd $NPB/bin #if current directory has changed
> less $SCRATCH/MAQAO_HANDSON/bt/OV_btmz_srun.json
```

```
“executable”: "bt-mz.C.x"
...
“number_nodes”: 2
”number_processes_per_node”: 2
...
“envv_OMP_NUM_THREADS”: 48
...
“mpi_command”: "srun -p standard96s --reservation=vi-hps-tuning
-N <number_nodes> --tasks-per-node=<number_processes_per_node> -c
<OMP_NUM_THREADS>"
```

(optional) Launch MAQAO ONE View on bt-mz (with config file)

Launch ONE View (with config)

```
> cd $NPB/bin
> maqao oneview -R1
--config=$SCRATCH/MAQAO_HANDSON/bt/OV_btmz_srun.json \
-xp=OV_btmz_srun --show-program-output
```

(optional) Setup ONE View (sbatch)

The ONE View configuration file must contain all variables for executing the application. Retrieve the configuration file prepared for bt-mz in batch mode from the MAQAO_HANDSON directory

```
> cd $NPB/bin
> less $SCRATCH/MAQAO_HANDSON/bt/OV_btmz_sbatch.json
```

```
“executable”: "bt-mz.C.x"
…
“scripts”: {
  “command”: "sbatch <myscript>",
  “files”: [
    {“path”: “maqao.sbatch”, “tag”: "myscript“}
…
“number_nodes”: 2
”number_processes_per_node”: 2
”envv_OMP_NUM_THREADS”: 48
…
“mpi_command”: "srun"
```

(optional) Review jobscript for use with ONE View

All variables in the jobscript defined in the configuration file must be replaced with their name from it.
Retrieve jobscript modified for ONE View from the MAQAO_HANDSON directory.

```
> cd $NPB/bin #if current directory has changed
> cp $SCRATCH/MAQAO_HANDSON/bt/maqao.sbatch .
> less maqao.sbatch
```

```
...
#SBATCH --nodes=2<number_nodes>
#SBATCH --ntasks-per-node=2<number_processes_per_node>
#SBATCH --cpus-per-task=48<OMP_NUM_THREADS>
...
export OMP_NUM_THREADS=48<OMP_NUM_THREADS>
...
srun ... $EXE
<mpi_command> <run_command>
...
```

(optional) Launch MAQAO ONE View on bt-mz (batch mode)

Launch ONE View

```
> cd $NPB/bin #if current directory has changed
> maqao oneview -R1
--config=$SCRATCH/MAQAO_HANDSON/bt/OV_btmz_sbatch.json -
xp=OV_btmz_sbatch
```

The -xp parameter allows to set the path to the experiment directory, where ONE View stores the analysis results and where the reports will be generated.
If -xp is omitted, the experiment directory will be named maqao_<timestamp>.
WARNINGS: If the directory specified with -xp already exists, ONE View will reuse its content but not overwrite it. To overwrite, use --replace

Scalability profiling of bt-mz with MAQAO

Cédric Valensi

Setup ONE View for scalability analysis

Retrieve the configuration file prepared for bt-mz in batch mode from the MAQAO_HANDSON directory

```
> cd $NPB/bin #if cur. dir. has changed
> less $SCRATCH/MAQAO_HANDSON/bt/OV_btmz_scal.json
```

```
“executable”: "./bt-mz.C.x"
…
“run_command”: "<executable>"
…
“scripts”: {
  “command”: "sbatch <myscript>",
  “files”: [
    {“path”: “maqao.sbatch”, “tag”: "myscript“}
…
“number_nodes”: 1
“number_processes_per_node”: 1
…
“envv_OMP_NUM_THREADS”: 48
…
“mpi_command”: “srun"
…
“multiruns_params”: [
 {name: “2x1x48”, “number_nodes”: 2, “number_processes_per_node”: 1},
 {name: “2x2x48”, “number_nodes”: 2, “number_processes_per_node”: 2},
]
“base_run_name”: “1x1x48"
```

Launch MAQAO ONE View on bt-mz (scalability mode)

Launch ONE View (execution will be longer!)

```
> maqao oneview -R1 --with-scalability=strong \
-c=$SCRATCH/MAQAO_HANDSON/bt/OV_btmz_scal.json -xp=OV_btmz_scal
```

The results can then be accessed similarly to the analysis report.

```
> firefox gwdg_NPB/bin/OV_btmz_scal/RESULTS/bt-mz.C.x_one_html/index.html
```

OR

```
> tar czf $SCRATCH/bt_scal.tgz OV_btmz_scal/RESULTS/bt-mz.C.x_one_html
```

```
> scp <user>@inti.ocre.cea.fr:/scratch/lustre-grete/projects/tr_20260831-
vi-hps-tuning/<username>/bt_scal.tgz .
> tar xf bt_scal.tgz
> firefox OV_btmz_scal/RESULTS/bt-mz.C.x_one_html/index.html
```

A sample result directory is in **`MAQAO_HANDSON/bt/offline.tgz`**

Optimising a code with MAQAO

Cédric Valensi / Emmanuel OSERET

Matrix Multiply code

```
void kernel0 (int n,
              float a[n][n],
              float b[n][n],
              float c[n][n]) {
  int i, j, k;

  for (i=0; i<n; i++)
    for (j=0; j<n; j++) {
      c[i][j] = 0.0f;
      for (k=0; k<n; k++)
        c[i][j] += a[i][k] * b[k][j];
    }
}
```

"Naïve" dense matrix multiply implementation in C

Compile with GNU compiler

Go to the handson directory

```
> cd $SCRATCH/MAQAO_HANDSON/matmul
```

Compile all variants

```
> module load gcc/14.2.0
> make all
```

Analysing matrix multiply with MAQAO

Parameters are: <size of matrix> <number of repetitions>

```
> srun --reservation=vi-hps-tuning -p standard96s:shared
./matmul_orig/matmul 400 500
cycles per inner loop iter.: 2.05
```

Analyse matrix multiply with ONE View

```
> maqao oneview -R1 mpi-command="srun --reservation=vi-hps-
tuning -p standard96s:shared" -xp=OV_orig --
./matmul_orig/matmul 400 500
```

OR

```
> maqao oneview -R1 -c=OV_orig.json -xp=OV_orig
```

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox gwdg_scratch/MAQAO_HANDSON/matmul/OV_orig/RESULTS/\
matmul_one_html/index.html &
```

Global Metrics	
Total Time (s)	27.25
Max (Thread Active Time) (s)	27.24
Average Active Time (s)	27.24
Activity Ratio (%)	100.0
Average number of active threads	1.000
Affinity Stability (%)	91.1
Time in analyzed loops (%)	100.0
Time in analyzed innermost loops (%)	99.9
Time in user code (%)	100
Compilation Options Score (%)	50.0
Array Access Efficiency (%)	83.3

Potential Speedups		
Perfect Flow Complexity		1.00
Perfect OpenMP/MPI/Pthread/TBB		1.00
Perfect OpenMP/MPI/Pthread/TBB + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.00
	Nb Loops to get 80%	1
FP Vectorised	Potential Speedup	2.81
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	16.0
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	1.00
	Nb Loops to get 80%	1

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox gwdg_scratch/MAQAO_HANDSON/matmul/OV_orig/RESULTS/\
matmul_one_html/index.html &
```

Loop id	Source Location	Source Function	Level	Max Thread Time / Walltime run_0 (%)	Exclusive Coverage run_0 (%)	Max Exclusive Time Over Threads run_0 (s)	Vectorization Ratio (%)	Vector Length Use (%)
1	matmul - kernel.c:24-25	kernel	Innermost	99.84	99.89	27.21	0	6.25
2	matmul - kernel.c:7-25 [...]	kernel	InBetween	0.11	0.11	0.03	33.33	14.58

Vectorization

Your loop is not vectorized. 16 data elements could be processed at once in vector registers.

By vectorizing your loop, you can lower the cost of an iteration from 3.00 to 0.19 cycles (16.00x speedup).

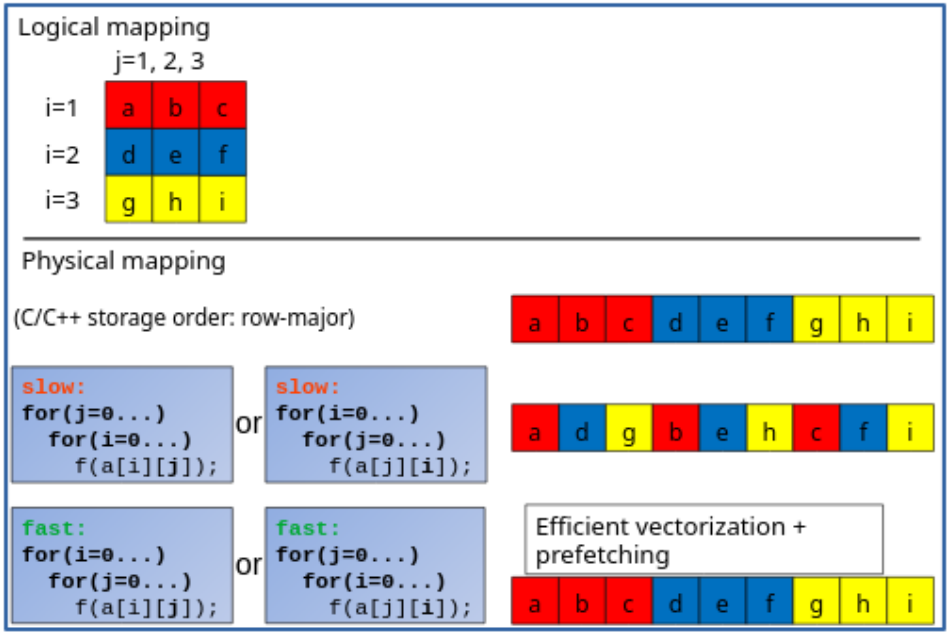
Details

All SSE/AVX instructions are used in scalar version (process only one data element in vector registers). Since your execution units are vector units, only a vectorized loop can use their full power.

CQA output for the baseline kernel

Workaround

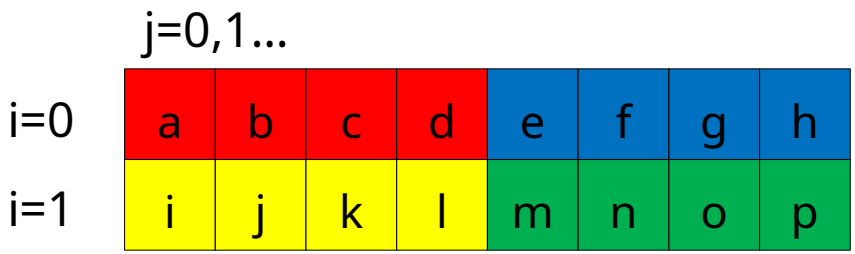
- Try another compiler or update/tune your current one:
  - recompile with fassociative-math (included in Ofast or ffast-math) to extend loop vectorization to FP reductions.
- Remove inter-iterations dependences from your loop and make it unit-stride:
  - If your arrays have 2 or more dimensions, check whether elements are accessed contiguously and, otherwise, try to permute loops accordingly: C storage order is row-major: for(i) for(j) a[j][i] = b[j][i]; (slow, non stride 1) => for(i) for(j) a[i][j] = b[i][j]; (fast, stride 1)



  - If your loop streams arrays of structures (AoS), try to use structures of arrays instead (SoA): for(i) a[i].x = b[i].x; (slow, non stride 1) => for(i) a.x[i] = b.x[i]; (fast, stride 1)

Impact of loop permutation on data access

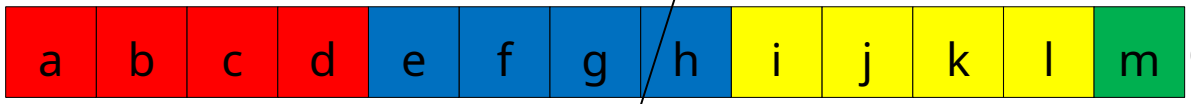
Logical mapping



Efficient vectorization + prefetching

Physical mapping

(C stor. order: row-major)




```
for (j=0; j<n; j++)
  for (i=0; i<n; i++)
    f(a[i][j]);
```


```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    f(a[i][j]);
```

Removing inter-iteration dependences and getting stride 1 by permuting loops on j and k

```
void kernel0 (int n,
              float a[n][n],
              float b[n][n],
              float c[n][n]) {
  int i, j, k;

  for (i=0; i<n; i++)
    for (j=0; j<n; j++) {
      c[i][j] = 0.0f;
      for (k=0; k<n; k++)
        c[i][j] += a[i][k] *
                   b[k][j];
    }
}
```

```
void kernel1 (int n,
              float a[n][n],
              float b[n][n],
              float c[n][n]) {
  int i, j, k;

  for (i=0; i<n; i++) {
    for (j=0; j<n; j++)
      c[i][j] = 0.0f;
    for (k=0; k<n; k++)
      for (j=0; j<n; j++)
        c[i][j] += a[i][k] *
                   b[k][j];
  }
}
```

Analyse matrix multiply with permuted loops

Run permuted loops version of matrix multiply

```
> srun --reservation=vi-hps-tuning -p standard96s:shared
./matmul_perm/matmul 400 500
cycles per inner loop iter.: 0.32
```

Analyse matrix multiply with ONE View

```
> maqao oneview -R1 -c=OV_perm.json -xp=OV_perm
```

OR

```
> maqao oneview -R1 mpi-command="srun --reservation=vi-hps-
tuning -p standard96s:shared" -xp=OV_perm --
./matmul_perm/matmul 400 500
```

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox gwdg_scratch/MAQAO_HANDSON/matmul/OV_perm/RESULTS/\
matmul_one_html/index.html &
```

Global Metrics	
Total Time (s)	4.38
Max (Thread Active Time) (s)	4.37
Average Active Time (s)	4.37
Activity Ratio (%)	99.9
Average number of active threads	0.999
Affinity Stability (%)	99.7
Time in analyzed loops (%)	100.0
Time in analyzed innermost loops (%)	96.6
Time in user code (%)	100
Compilation Options Score (%)	50.0
Array Access Efficiency (%)	100

Faster (was 27.25)

Potential Speedups		
Perfect Flow Complexity		1.00
Perfect OpenMP/MPI/Pthread/TBB		1.00
Perfect OpenMP/MPI/Pthread/TBB + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.02
	Nb Loops to get 80%	1
FP Vectorised	Potential Speedup	1.77
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	4.10
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	1.19
	Nb Loops to get 80%	1

More efficient vecto. (was 16.00)

Viewing results (HTML)

Loop id	Source Location	Source Function	Level	Max Thread Time / Walltime run_0 (%)	Exclusive Coverage run_0 (%)	Max Exclusive Time Over Threads run_0 (s)	Vectorization Ratio (%)	Vector Length Use (%)
5	matmul - kernel.c:24-25	kernel	Innermost	96.43	96.57	4.22	100	25
4	matmul - kernel.c:3-25 [...]	kernel	InBetween	3.31	3.32	0.14	20	11.25

Vectorization

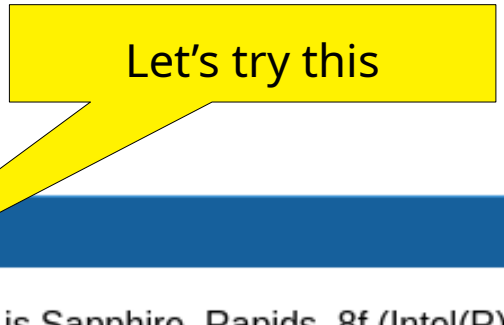
Your loop is vectorized, but using only 128 out of 512 bits (SSE/AVX-128 instructions on AVX-512 processors).

By fully vectorizing your loop, you can lower the cost of an iteration from 1.17 to 0.29 cycles (4.00x speedup).

Details

All SSE/AVX instructions are used in vector version (process two or more data elements in vector registers). Since your execution units are vector units, only a fully vectorized loop can use their full power.

Viewing results (HTML)



Workaround

- Recompile with march=sapphirerapids. CQA target is Sapphire_Rapids_8f (Intel(R) Xeon(R) Sapphire Rapids) but specialization flags are -march=x86-64
- Use vector aligned instructions:
  1. align your arrays on 64 bytes boundaries: replace { void *p = malloc (size); } with { void *p; posix_memalign (&p, 64, size); }.
  2. inform your compiler that your arrays are vector aligned: if array 'foo' is 64 bytes-aligned, define a pointer 'p_foo' as __builtin_assume_aligned (foo, 64) and use it instead of 'foo' in the loop.

Viewing results (HTML)

Global Metrics	
Total Time (s)	4.38
Max (Thread Active Time) (s)	4.37
Average Active Time (s)	4.37
Activity Ratio (%)	99.9
Average number of active threads	0.999
Affinity Stability (%)	99.7
Time in analyzed loops (%)	100.0
Time in analyzed innermost loops (%)	96.6
Time in user code (%)	100
Compilation Options Score (%)	50.0
Array Access Efficiency (%)	100

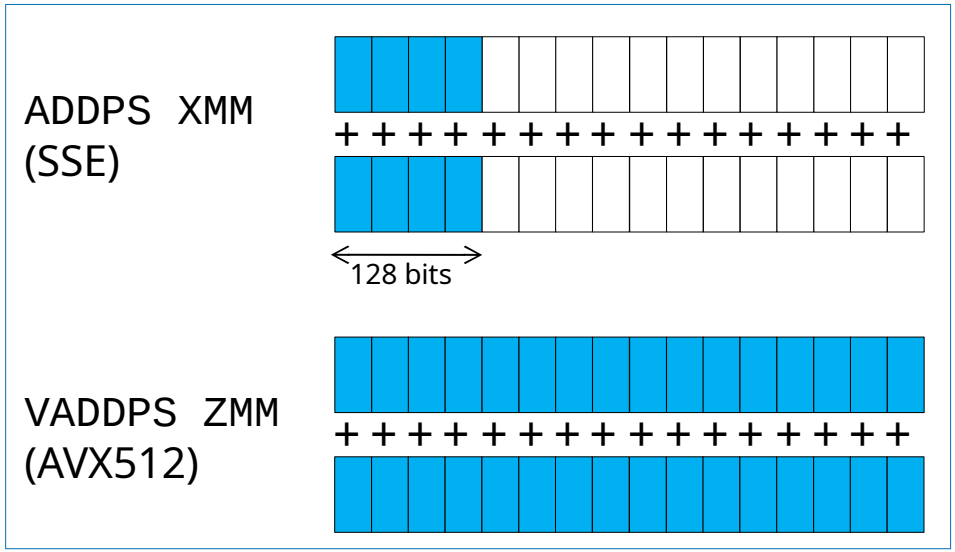
Source Object	Issue
▼ matmul	
▼ kernel.c	
○	-march=x86-64 is used but it should be replaced by a more architecture specific option or -march=native.
○	-funroll-loops is missing.

Let’s try this

Impacts of architecture specialization: vectorization

- Vectorization
  - SSE instructions (SIMD 128 bits) used on a processor supporting AVX512 ones (SIMD 512 bits)
  - => 75% efficiency loss

- FMA: a + b x c with cost of b x c



Analyse matrix multiply with compiler optimizations (microarchitecture-specialization and loop unrolling)

Run array-aligned version of matrix multiply

```
> srun --reservation=vi-hps-tuning -p standard96s:shared
./matmul_comp_opt/matmul 400 500
cycles per FMA: 0.29
```

Analyse matrix multiply with ONE View

```
> maqao oneview -R1 mpi-command="srun --reservation=vi-hps-tuning -p
standard96s:shared" -xp=OV_comp_opt -- ./matmul_comp_opt/matmul 400 500
```

OR

```
> maqao oneview -R1 -c=OV_comp_opt.json -xp=OV_comp_opt
```

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox gwdg_scratch/MAQAO_HANDSON/matmul/OV_comp_opt/RESULTS/\
matmul_one_html/index.html &
```

Global Metrics

Total Time (s)	3.83
Max (Thread Active Time) (s)	3.82
Average Active Time (s)	3.82
Activity Ratio (%)	99.9
Average number of active threads	0.999
Affinity Stability (%)	99.8
Time in analyzed loops (%)	100.0
Time in analyzed innermost loops (%)	92.8
Time in user code (%)	100
Compilation Options Score (%)	100
Array Access Efficiency (%)	81.3

Faster (was 4.38)

Potential Speedups

Perfect Flow Complexity		1.00
Perfect OpenMP/MPI/Pthread/TBB		1.00
Perfect OpenMP/MPI/Pthread/TBB + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.04
	Nb Loops to get 80%	1
FP Vectorised	Potential Speedup	1.11
	Nb Loops to get 80%	2
Fully Vectorised	Potential Speedup	2.11
	Nb Loops to get 80%	2
FP Arithmetic Only	Potential Speedup	1.46
	Nb Loops to get 8	1

2x better (was 4.10) but still 50 % efficiency loss

Viewing results (HTML)

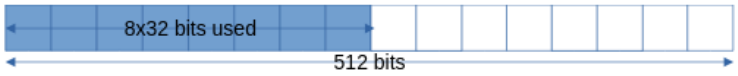
On your local machine (sshfs):

```
> firefox gwdg_scratch/MAQAO_HANDSON/matmul/OV_comp_opt/RESULTS/\
matmul_one_html/index.html &
```

Loop id	Source Location	Source Function	Level	Max Thread Time / Walltime run_0 (%)	Exclusive Coverage run_0 (%)	Max Exclusive Time Over Threads run_0 (s)	Vectorization Ratio (%)	Vector Length Use (%)
5	matmul - kernel.c:24-25	kernel	Innermost	92.76	92.81	3.55	100	50
4	matmul - kernel.c:3-25 [...]	kernel	InBetween	7.19	7.19	0.28	44.44	25.58

Vectorization

Your loop is vectorized, but using only 256 out of 512 bits (AVX/AVX2 instructions on AVX-512 processors).



Details

All SSE/AVX instructions are used in vector version (process two or more data elements in vector registers).

512-bits vectorization

On some x86 processors supporting 512-bits vectorization, compilers are often too conservative and limit vectorization to 256 bits. Performance can then be improved by enforcing 512-bits vectorization, especially with many vectorized and high trip count loops. 512-bits vectorization performance overhead (compared to 256-bits) is generally lower on newer processors.

Workaround

Recompile with -mprefer-vector-width=512

Let's try this

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox gwdg_scratch/MAQAO_HANDSON/matmul/OV_comp_opt/RESULTS/\
matmul_one_html/index.html &
```

Vector unaligned load/store instructions

Detected 16 optimal vector unaligned load/store instructions.

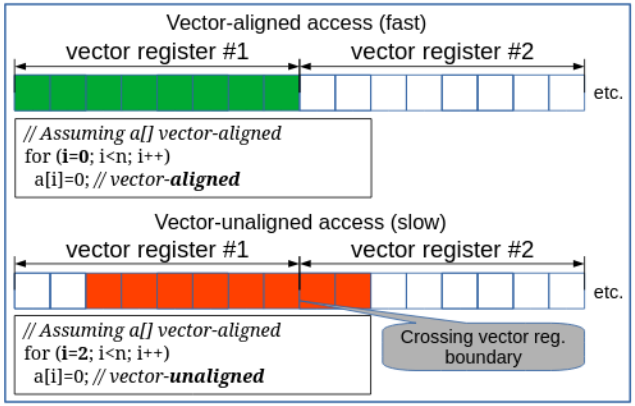
Details

- VMOVUPS: 16 occurrences ▶

Workaround

Use vector aligned instructions:

1. align your arrays on 32 bytes boundaries: replace { void *p = malloc (size); } with { void *p; posix_memalign (&p, 32, size); }.
2. inform your compiler that your arrays are vector aligned: if array 'foo' is 32 bytes-aligned, define a pointer 'p_foo' as __builtin_assume_aligned (foo, 32) and use it instead of 'foo' in the loop.



Analyse matrix multiply with 512 bits vectorization

Run array-aligned version of matrix multiply

```
> srun --reservation=vi-hps-tuning -p standard96s:shared
./matmul_vec512/matmul 400 500
cycles per FMA: 0.26
```

Analyse matrix multiply with ONE View

```
> maqao oneview -R1 mpi-command="srun --reservation=vi-hps-tuning -p
standard96s:shared" -xp=OV_vec512 -- ./matmul_vec512/matmul 400 500
```

OR

```
> maqao oneview -R1 -c=OV_vec512.json -xp=OV_vec512
```

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox gwdg_scratch/MAQAO_HANDSON/matmul/OV_comp_opt/RESULTS/\
matmul_one_html/index.html &
```

Global Metrics

Total Time (s)	3.42
Max (Thread Active Time) (s)	3.41
Average Active Time (s)	3.41
Activity Ratio (%)	99.8
Average number of active threads	0.998
Affinity Stability (%)	99.7
Time in analyzed loops (%)	99.9
Time in analyzed innermost loops (%)	89.0
Time in user code (%)	99.8
Compilation Options Score (%)	100
Array Access Efficiency (%)	81.3

Faster (was 3.83)

Potential Speedups

Perfect Flow Complexity		1.00
Perfect OpenMP/MPI/Pthread/TBB		1.00
Perfect OpenMP/MPI/Pthread/TBB + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.05
	Nb Loops to get 80%	1
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	1.03
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	1.48
	Nb Loops to get 8	2

Now optimal vectorization (was 2.11)

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox gwdg_scratch/MAQAO_HANDSON/matmul/OV_comp_opt/RESULTS/\
matmul_one_html/index.html &
```

Loop id	Source Location	Source Function	Level	Max Thread Time / Walltime run_0 (%)	Exclusive Coverage run_0 (%)	Max Exclusive Time Over Threads run_0 (s)	Vectorization Ratio (%)	Vector Length Use (%)
5	matmul - kernel.c:24-25	kernel	Innermost	88.87	89.02	3.04	100	100
4	matmul - kernel.c:3-25 [...]	kernel	InBetween	10.82	10.83	0.37	32.43	35.56

Vectorization

Your loop is fully vectorized, using full register length.

Details

All SSE/AVX instructions are used in vector version (process two or more data elements in vector registers).

Viewing results (HTML) (reminder, displayed for the comp-opt version)

On your local machine (sshfs):

```
> firefox gwdg_scratch/MAQAO_HANDSON/matmul/OV_comp_opt/RESULTS/\
matmul_one_html/index.html &
```

Vector unaligned load/store instructions

Detected 16 optimal vector unaligned load/store instructions.

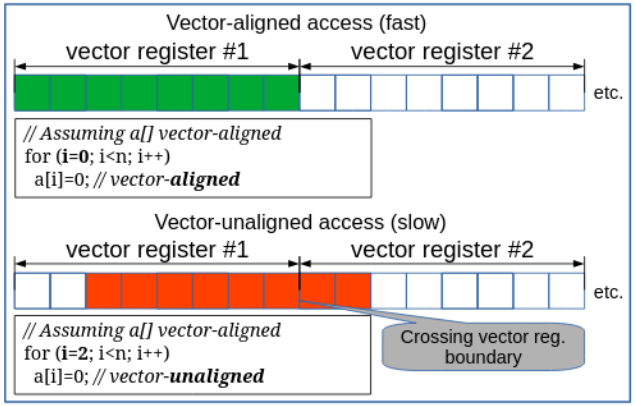
Details

- VMOVUPS: 16 occurrences ▶

Workaround

Use vector aligned instructions:

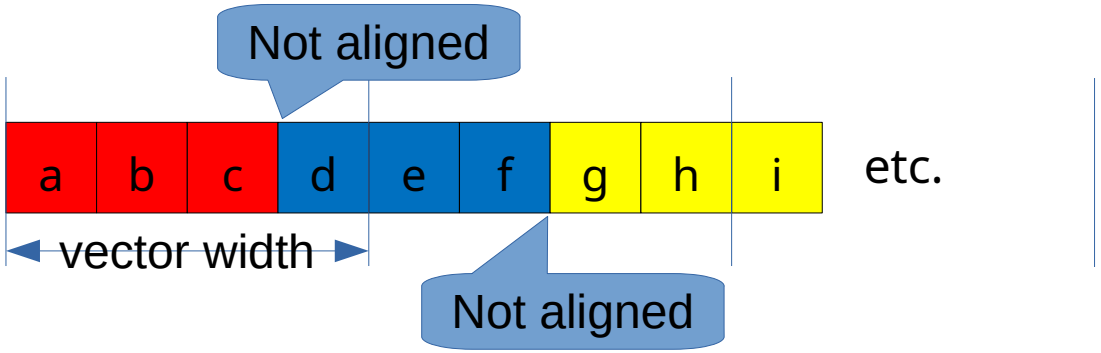
1. align your arrays on 32 bytes boundaries: replace { void *p = malloc (size); } with { void *p; posix_memalign (&p, 32, size); }.
2. inform your compiler that your arrays are vector aligned: if array 'foo' is 32 bytes-aligned, define a pointer 'p_foo' as __builtin_assume_aligned (foo, 32) and use it instead of 'foo' in the loop.



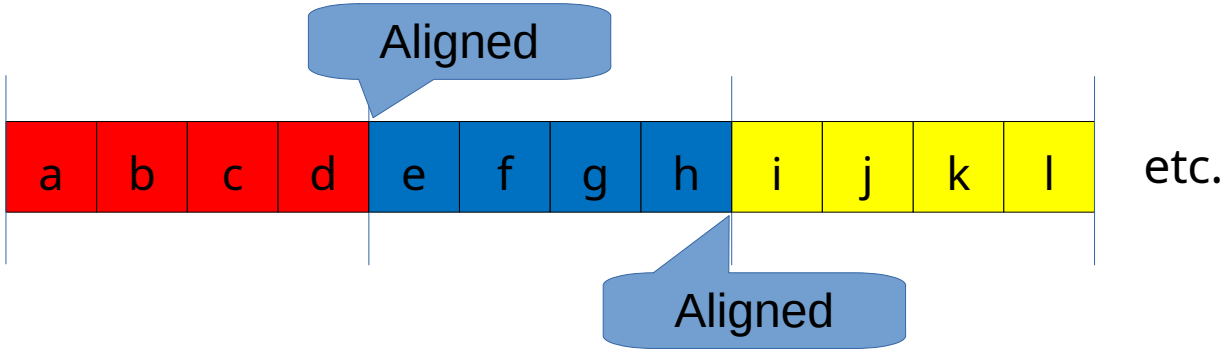
Multidimensional array alignment

Data organized as a 2D array: n lines of 3 columns
Each vector can hold 4 consecutive elements

a[n][3], only 1st element is aligned



a[n][4], 1st element of each line are aligned



Analyse matrix multiply with array alignment

Run unrolled version of matrix multiply

```
> srun --reservation=vi-hps-tuning -p standard96s:shared
./matmul_align/matmul 400 500
driver.c: Using posix_memalign instead of malloc
cycles per FMA: 0.13
```

Analyse matrix multiply with ONE View

```
> maqao oneview -R1 mpi-command="srun --reservation=vi-hps-
tuning -p standard96s:shared" -xp=OV_align --
./matmul_align/matmul 400 500
```

OR

```
> maqao oneview -R1 -c=OV_align.json -xp=OV_align
```

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox gwdg_scratch/MAQAO_HANDSON/matmul/OV_align/RESULTS/\
matmul_one_html/index.html &
```

Global Metrics

Total Time (s)	1.78
Max (Thread Active Time) (s)	1.78
Average Active Time (s)	1.78
Activity Ratio (%)	99.9
Average number of active threads	0.999
Affinity Stability (%)	99.6
Time in analyzed loops (%)	100.0
Time in analyzed innermost loops (%)	94.4
Time in user code (%)	100
Compilation Options Score (%)	100
Array Access Efficiency (%)	100

2x (was 3.42)

Using comparison mode: global level

```
> maqao oneview --compare-reports -xp=OV_cmp \
-inputs=OV_orig,OV_perm,OV_comp_opt,OV_vec512,OV_align
```

▼ **Compared Reports**

- r0: OV_orig
- r1: OV_perm
- r2: OV_comp_opt
- r3: OV_vec512
- r4: OV_align

Remark: open OV_cmp/RESULTS/OV_cmp/index.html

Global Metrics

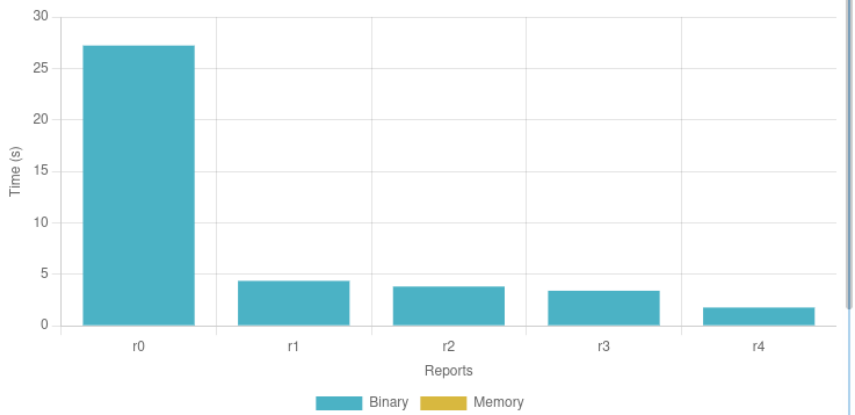
Metric	r0	r1	r2	r3	r4
Total Time (s)	27.25	4.38	3.83	3.42	1.78
Max (Thread Active Time) (s)	27.24	4.37	3.82	3.41	1.78
Average Active Time (s)	27.24	4.37	3.82	3.41	1.78
Activity Ratio (%)	100.0	99.9	99.9	99.8	99.9
Average number of active threads	1.000	0.999	0.999	0.998	0.999
Affinity Stability (%)	91.1	99.7	99.8	99.7	99.6
Time in analyzed loops (%)	100.0	100.0	100.0	99.9	100.0
Time in analyzed innermost loops (%)	99.9	96.6	92.8	89.0	94.4
Time in user code (%)	100	100	100	99.8	100
Compilation Options Score (%)	50.0	50.0	100	100	100
Array Access Efficiency (%)	83.3	100	81.3	81.3	100

Potential Speedups

		r0	r1	r2	r3	r4
Perfect Flow Complexity		1.00	1.00	1.00	1.00	1.00
Perfect OpenMP/MPI/Pthread/TBB		1.00	1.00	1.00	1.00	1.00
Perfect OpenMP/MPI/Pthread/TBB + Perfect Load Distribution		1.00	1.00	1.00	1.00	1.00
No Scalar Integer	Potential Speedup	1.00	1.02	1.04	1.05	1.02
	Nb Loops to get 80%	1	1	1	1	1
FP Vectorised	Potential Speedup	2.81	1.77	1.11	1.00	1.00
	Nb Loops to get 80%	1	1	2	1	1
Fully Vectorised	Potential Speedup	16.0	4.10	2.11	1.03	1.02
	Nb Loops to get 80%	1	1	2	1	1
Only FP Arithmetic	Potential Speedup	1.00	1.19	1.46	1.48	1.45
	Nb Loops to get 80%	1	1	1	2	1

Application Categorization

Time



Using comparison mode: experiment summaries

Experiment Summaries

	r0	r1	r2	r3	r4
Experiment Name					
Application	./matmul_orig/matmul	./matmul_perm/matmul	./matmul_comp_opt/matmul	./matmul_vec512/matmul	./matmul_align/matmul
Timestamp	2026-06-19 15:27:27	2026-06-19 15:28:44	2026-06-19 15:30:26	2026-06-19 15:43:35	2026-06-19 15:46:36
Experiment Type	Sequential	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Machine	c0433	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Architecture	x86_64	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Micro Architecture	SAPPHIRE_RAPIDS	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Model Name	Intel(R) Xeon(R) Platinum 8468	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Cache Size	107520 KB	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Number of Cores	48	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Maximal Frequency	3.80 GHz	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
OS Version	Linux 4.18.0-553.124.1.el8_10.0.2.x86_64 #1 SMP Mon May 18 18:00:46 UTC 2026	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Architecture used during static analysis	x86_64	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Micro Architecture used during static analysis	SAPPHIRE_RAPIDS	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Compilation Options	**matmul**: GNU C17 14.2.0 -mtune=generic -march=x86-64 -g -O3 -fno-omit-frame-pointer	*same as **r0***	**matmul**: GNU C17 14.2.0 -march=sapphirerapids -g -O3 -funroll-loops -fno-omit-frame-pointer	**matmul**: GNU C17 14.2.0 -march=sapphirerapids -mprefer-vector-width=512 -g -O3 -funroll-loops -fno-omit-frame-pointer	*same as **r3***
Number of processes observed	1	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Number of threads observed	1	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Frequency Driver	intel_cpufreq	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Frequency Governor	performance	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Huge Pages	always	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Hyperthreading	on	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Number of sockets	2	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
Number of cores per socket	48	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
MAQAO version	2026.0.1	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***
MAQAO build	68c1d977d600f5f7f9a93ba9fba47128e0e19aa7::20260619-112330	*same as **r0***	*same as **r0***	*same as **r0***	*same as **r0***

Using comparison mode: experiment summaries

▼ Experiment Quality

OV_orig	OV_perm	OV_comp_opt	OV_vec512	OV_align
[2 / 3] Security settings from the host restrict profiling. Some metrics will be missing or incomplete. Current value for kernel.perf_event_paranoid is 2. If possible, set it to 1 or check with your system administrator which flag can be used to achieve this.	[2 / 3] Security settings from the host restrict profiling. Some metrics will be missing or incomplete. Current value for kernel.perf_event_paranoid is 2. If possible, set it to 1 or check with your system administrator which flag can be used to achieve this.	[2 / 3] Security settings from the host restrict profiling. Some metrics will be missing or incomplete. Current value for kernel.perf_event_paranoid is 2. If possible, set it to 1 or check with your system administrator which flag can be used to achieve this.	[2 / 3] Security settings from the host restrict profiling. Some metrics will be missing or incomplete. Current value for kernel.perf_event_paranoid is 2. If possible, set it to 1 or check with your system administrator which flag can be used to achieve this.	[2 / 3] Security settings from the host restrict profiling. Some metrics will be missing or incomplete. Current value for kernel.perf_event_paranoid is 2. If possible, set it to 1 or check with your system administrator which flag can be used to achieve this.
[3 / 3] Most of time spent in analyzed modules comes from functions with source/debug info -g option gives access to debugging informations, such are source locations.	[3 / 3] Most of time spent in analyzed modules comes from functions with source/debug info -g option gives access to debugging informations, such are source locations.	[3 / 3] Most of time spent in analyzed modules comes from functions with source/debug info -g option gives access to debugging informations, such are source locations.	[3 / 3] Most of time spent in analyzed modules comes from functions with source/debug info -g option gives access to debugging informations, such are source locations.	[3 / 3] Most of time spent in analyzed modules comes from functions with source/debug info -g option gives access to debugging informations, such are source locations.
[3 / 3] Most of time spent in analyzed modules comes from functions with compilation options informations and -fno-omit-frame-pointer is present -fno-omit-frame-pointer improves the accuracy of callchains found during the application profiling.	[3 / 3] Most of time spent in analyzed modules comes from functions with compilation options informations and -fno-omit-frame-pointer is present -fno-omit-frame-pointer improves the accuracy of callchains found during the application profiling.	[3 / 3] Most of time spent in analyzed modules comes from functions with compilation options informations and -fno-omit-frame-pointer is present -fno-omit-frame-pointer improves the accuracy of callchains found during the application profiling.	[3 / 3] Most of time spent in analyzed modules comes from functions with compilation options informations and -fno-omit-frame-pointer is present -fno-omit-frame-pointer improves the accuracy of callchains found during the application profiling.	[3 / 3] Most of time spent in analyzed modules comes from functions with compilation options informations and -fno-omit-frame-pointer is present -fno-omit-frame-pointer improves the accuracy of callchains found during the application profiling.
[2 / 2] Application is correctly profiled ("Others" category represents 0.00 % of the execution time)	[2 / 2] Application is correctly profiled ("Others" category represents 0.00 % of the execution time)	[2 / 2] Application is correctly profiled ("Others" category represents 0.00 % of the execution time)	[2 / 2] Application is correctly profiled ("Others" category represents 0.00 % of the execution time)	[2 / 2] Application is correctly profiled ("Others" category represents 0.00 % of the execution time)

▼ Loops Overview

▶ Loops List

	Analysis	r0	r1	r2	r3	r4
Loop Computation Issues	Less than 10% of the FP ADD/SUB/MUL arithmetic operations are performed using FMA	1	2	0	0	0
	Presence of a large number of scalar integer instructions	1	1	1	1	1
Control Flow Issues	Presence of calls	0	1	0	0	0
	Presence of more than 4 paths	0	2	1	1	1
	Non-innermost loop	1	2	1	1	1
Data Access Issues	Presence of constant non-unit stride data access	2	0	1	1	0
	More than 10% of the vector loads instructions are unaligned	0	1	2	1	0
	Presence of special instructions executing on a single port	0	1	1	1	1
	More than 20% of the loads are accessing the stack	0	2	1	0	0
Vectorization Roadblocks	Presence of calls	0	1	0	0	0
	Presence of more than 4 paths	0	2	1	1	1
	Non-innermost loop	1	2	1	1	1
	Presence of constant non-unit stride data access	2	0	1	1	0
Inefficient Vectorization	Presence of special instructions executing on a single port	0	1	1	1	1

Using comparison mode: function & loop level

Functions

Name	Module	Coverage (%)					Inclusive Time w.r.t. Wall Time(s)					Max Inc. Time over Threads(s)						
		OV_orig	OV_perm	OV_comp_opt	OV_vec512	OV_align	OV_orig	OV_perm	OV_comp_opt	OV_vec512	OV_align	OV_orig	OV_perm	OV_comp_opt	OV_vec512	OV_align	OV_orig	OV_p
kernel	binary	100.00	100.00	100.00	99.85	100.00	27.24	4.37	3.82	3.41	1.78	27.24	4.37	3.82	3.41	1.78	1	1
__memset_avx512_unaligned_erms	libc-2.28.so	NA	NA	NA	0.15	NA	NA	NA	NA	0.00	NA	NA	NA	NA	0.00	NA	NA	NA

Loops

▼ **kernel.c: 24 - 472.67 %**

Run OV_orig

Loop Source Regions: • /user/emmanuel.oseret/u28863/MAQAO_HANDSON/matmul/matmul_orig/kernel.c: 24-25

ASM Loop ID	Max Time Over Threads (s)	Time w.r.t. Wall Time (s)	Cov (%)	Vect. Ratio (%)	Vector Length Use (%)
1	27.21	27.21	99.89	0	6.25

Sum on 1 analyzed binary loop (matmul - 1)

Analysis	Count
Loop Computation Issues	
Less than 10% of the FP ADD/SUB/MUL arithmetic operations are performed using FMA	1
Data Access Issues	
Presence of constant non-unit stride data access	1
More than 10% of the vector loads instructions are unaligned	0
Vectorization Roadblocks	
Presence of constant non-unit stride data access	1

Run OV_perm

Loop Source Regions: • /user/emmanuel.oseret/u28863/MAQAO_HANDSON/matmul/matmul_perm/kernel.c: 24-25

ASM Loop ID	Max Time Over Threads (s)	Time w.r.t. Wall Time (s)	Cov (%)	Vect. Ratio (%)	Vector Length Use (%)
5	4.22	4.22	96.57	100	25

Sum on 1 analyzed binary loop (matmul - 5)

Analysis	Count
Loop Computation Issues	
Less than 10% of the FP ADD/SUB/MUL arithmetic operations are performed using FMA	1
Data Access Issues	
Presence of constant non-unit stride data access	0
More than 10% of the vector loads instructions are unaligned	1
Vectorization Roadblocks	
Presence of constant non-unit stride data access	

Run OV_comp_opt

Loop Source Regions: • /user/emmanuel.oseret/u28863/MAQAO_HANDSON/matmul/matmul_comp_opt/kernel.c: 24-25

ASM Loop ID	Max Time Over Threads (s)	Time w.r.t. Wall Time (s)	Cov (%)	Vect. Ratio (%)	Vector Length Use (%)
5	3.55	3.55	92.81	100	50

Sum on 1 analyzed binary loop (matmul - 5)

Analysis	Count
Loop Computation Issues	
Less than 10% of the FP ADD/SUB/MUL arithmetic operations are performed using FMA	
Data Access Issues	
Presence of constant non-unit stride data access	1
More than 10% of the vector loads instructions are unaligned	1
Vectorization Roadblocks	
Presence of constant non-unit stride data access	1

Run OV_vec512

Loop Source Regions: • /user/emmanuel.oseret/u28863/MAQAO_HANDSON/matmul/matmul_vec512/kernel.c: 24-25

ASM Loop ID	Max Time Over Threads (s)	Time w.r.t. Wall Time (s)	Cov (%)	Vect. Ratio (%)	Vector Length Use (%)
5	3.04	3.04	89.02	100	100

Sum on 1 analyzed binary loop (matmul - 5)

Analysis	Count
Loop Computation Issues	
Less than 10% of the FP ADD/SUB/MUL arithmetic operations are performed using FMA	
Data Access Issues	
Presence of constant non-unit stride data access	1
More than 10% of the vector loads instructions are unaligned	0
Vectorization Roadblocks	
Presence of constant non-unit stride data access	1

Run OV_align

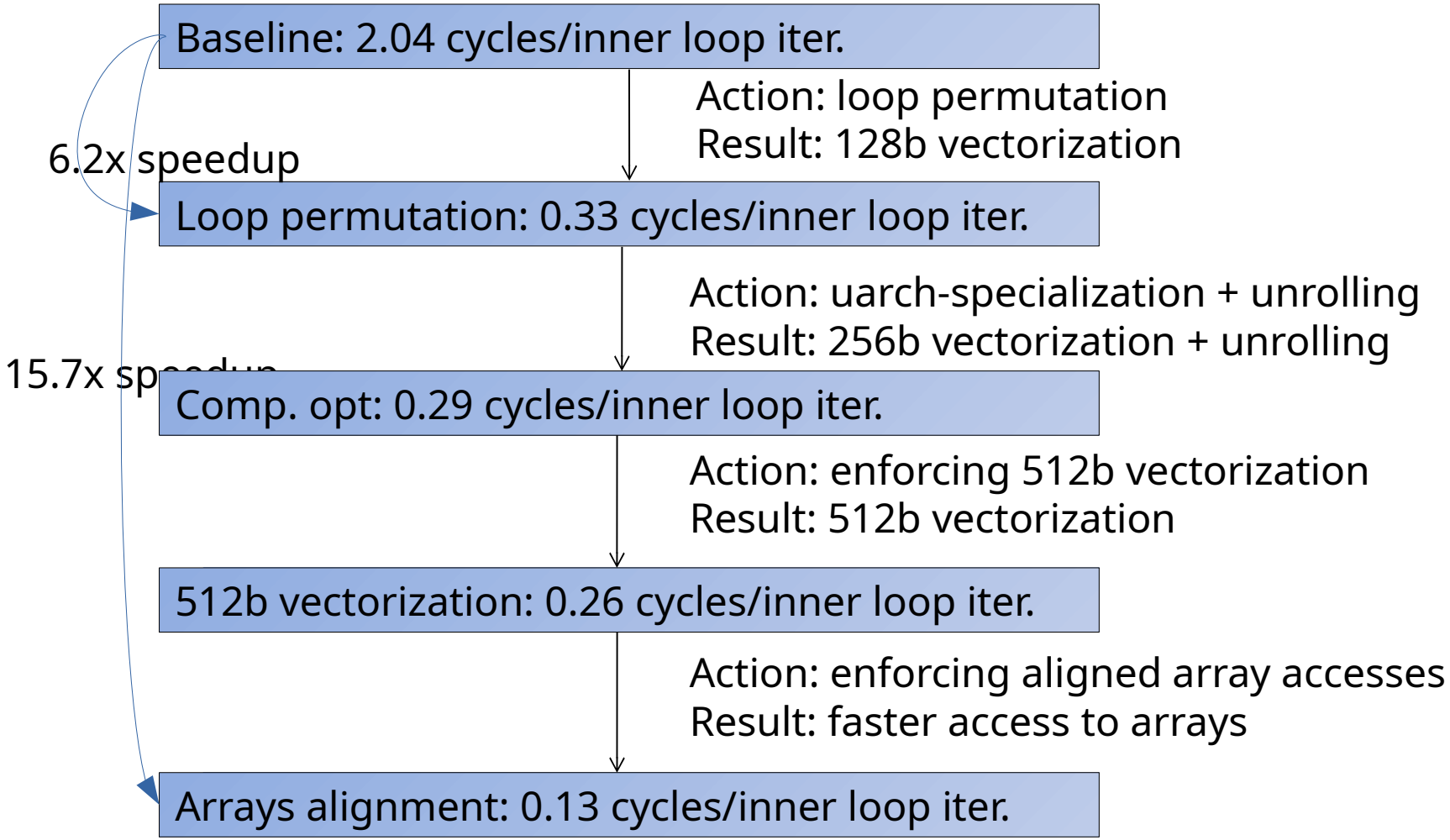
Loop Source Regions: • /user/emmanuel.oseret/u28863/MAQAO_HANDSON/matmul/matmul_align/kernel.c: 24-25

ASM Loop ID	Max Time Over Threads (s)	Time w.r.t. Wall Time (s)	Cov (%)	Vect. Ratio (%)	Vector Length Use (%)
5	1.68	1.68	94.38	100	100

Sum on 1 analyzed binary loop (matmul - 5)

Analysis	Count
Loop Computation Issues	
Less than 10% of the FP ADD/SUB/MUL arithmetic operations are performed using FMA	
Data Access Issues	
Presence of constant non-unit stride data access	
More than 10% of the vector loads instructions are unaligned	
Vectorization Roadblocks	
Presence of constant non-unit stride data access	

Summary of optimizations and gains



More sample codes

More codes to study with MAQAO in

```
$SCRATCH/MAQAO_HANDSON/loop_optim_tutorial.tgz
```