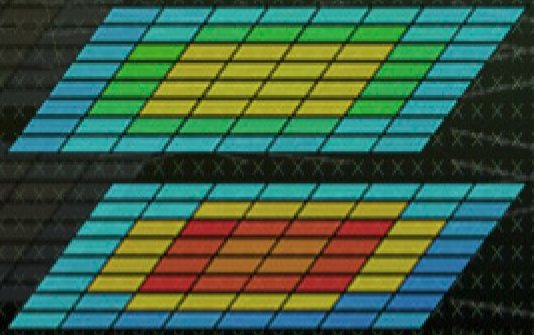


MAQAO
Hands-on exercises

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

Setup (reminder)

Login to the cluster

```
> ssh <username>@turpanlogin.calmip.univ-toulouse.fr
```

Copy handson material to your workspace directory

```
> cd $SCRATCH
> tar xvf /tmpdir/vi-hps/material/handsons/MAQAO_HANDSON.tgz
> tar xvf /tmpdir/vi-hps/material/handsons/NPB3.4-MZ-MPI.tgz
```

Load MAQAO environment

```
> module use /tmpdir/vi-hps/opt/modules
> module load maqao/2.19.0
```

Setup (bt-mz compilation with Intel compiler and MPI & debug symbols)

Go to the NPB directory provided with MAQAO handsons

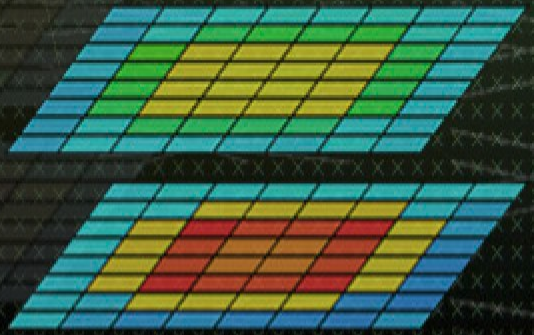
```
> cd $SCRATCH/NPB3.4-MZ-MPI
```

Load compiler and environment (if not already loaded)

```
> module load gnu/11.2.0
> module load openmpi/gnu/4.1.4-cpu
```

Compile and run

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



Profiling bt-mz with MAQAO

Cédric Valensi

Setup ONE View for batch mode

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 $SCRATCH/NPB3.4-MZ-MPI/bin
> cp $SCRATCH/MAQAO_HANDSON/bt/bt_oneview.json .
> less bt_oneview.json
```

```
"executable": "bt-mz.C.x"
"batch_script": "bt_maqao.sbatch"
"batch_command": "sbatch <batch_script>“
"number_processes": 16
"number_processes_per_node": 8
"number_nodes": 2
"envv_OMP_NUM_THREADS": 4
"mpi_command": "mpirun --bind-to none -n <number_processes>"
```

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 $SCRATCH/NPB3.4-MZ-MPI/bin #if current directory has changed
> cp $SCRATCH/MAQAO_HANDSON/bt/bt_maqao.sbatch .
> less bt_maqao.sbatch
```

```
...
#SBATCH -N=2<number_nodes>
#SBATCH -n=16<number_processes>
#SBATCH --ntasks-per-node=8<number_processes_per_node>
...
export OMP_NUM_THREADS=4<OMP_NUM_THREADS>
...
mpirun --bind-do-none ./bt-mz.C.x
<mpi_command> <run_command>
...
```

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

Launch ONE View

```
> cd $SCRATCH/NPB3.4-MZ-MPI/bin #if current directory has changed
> maqao oneview -R1 --config=bt_oneview.json -xp=ov_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.

Display MAQAO ONE View results

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

Mount $SCRATCH locally:

```
> mkdir turpan_work
> sshfs <user>@turpanlogin.calmip.univ-toulouse.fr:/tmpdir/<project>/<user> \
turpan_work
> firefox turpan_work/NPB3.4-MZ-MPI/bin/ov_sbatch/RESULTS/bt-mz.C.x_one_html/
index.html
```

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

```
> tar czf $HOME/bt_html.tgz ov_sbatch/RESULTS/bt-mz.C.x_one_html
```

```
> scp <user>@turpanlogin.calmip.univ-toulouse.fr:bt_html.tgz .
> tar xf bt_html.tgz
> firefox ov_sbatch/RESULTS/bt-mz.C.x_one_html/index.html
```

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
```

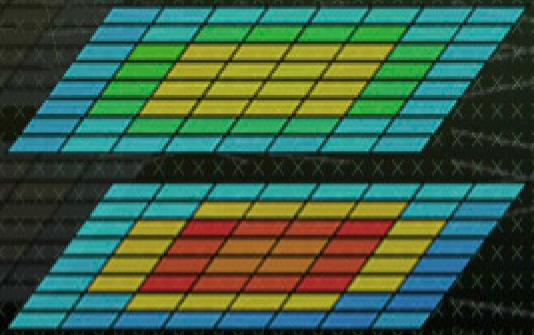
- scp is slow to copy directories (especially when containing many small files), copy a .tgz archive of the directory

Display MAQAO ONE View results as text (optional)

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

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

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



Scalability profiling of bt-mz with MAQAO

Cédric Valensi

Setup ONE View for scalability analysis

The configuration file contains parameters for executing a scalability analysis. They are stored inside an array containing all variables differing from the main run.

> less bt_oneview.json

```
"executable": "bt-mz.C.x"
"batch_script": "bt_maqao.sbatch"
"batch_command": "sbatch <batch_script>"
"number_processes": 16
"number_processes_per_node": 8
"number_nodes": 2
"envv_OMP_NUM_THREADS": 4
"mpi_command": "mpirun –bind-to-none -n <number_processes>"
"multiruns_params" = [
 {"name": "8P_2N", "number_processes": 8, "number_processes_per_node": 4, "number_nodes": 2},
 {"name": "8P_1N", "number_processes": 8, "number_processes_per_node": 8, "number_nodes": 1},
 {"name": "16P_1N", "number_processes": 16, "number_processes_per_node": 16, "number_nodes": 1}
]
"scalability_reference": "lowest-threads"
"base_run_name": "16P_2N"
```

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

Launch ONE View (execution will be longer!)

```
> maqao oneview -R1 --with-scalability=strong \
-c=bt_oneview.json -xp=ov_scal
```

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

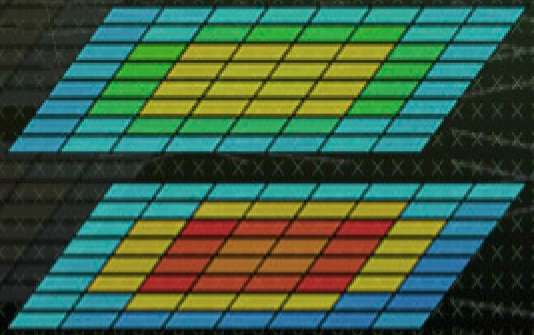
```
> firefox turpan_work/NPB3.4-MZ-MPI/bin/ov_scal/RESULTS/bt-
mz.C.x_one_html/index.html
```

OR

```
> tar czf $HOME/bt_scal.tgz \
ov_scal/RESULTS/bt-mz.C.x_one_html
```

```
> scp <user>@turpanlogin.calmip.univ-toulouse.fr:ov_scal.tgz .
> tar xf ov_scal.tgz
> firefox ov_scal/RESULTS/bt-mz.C.x_one_html/index.html
```

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



Optimising a code with MAQAO

Emmanuel OSERET
Hugo BOLLORÉ

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

Setup environment

Go to the handson directory

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

Load compiler environment

```
> module load gnu/11.2.0
```

Load MAQAO environment

```
> module use /tmpdir/vi-hps/opt/modules
> module load maqao
```

Analyse matrix multiply with MAQAO

Compile naïve implementation

```
> cd $SCRATCH/MAQAO_HANDSON/matmul #if cur. directory has changed
> make matmul_orig
```

Parameters: <matrix size> <number of repetitions>

```
> srun --reservation=VI-HPS matmul_orig/matmul 400 500
0.72 nanosecond per inner-loop iteration
```

Analyze with maqao

```
> maqao oneview -R1 -xp=ov_orig \
--mpi-command="srun --reservation=VI-HPS" \
-- matmul_orig/matmul 400 500
```

(alternate) Using configuration file for analysing matmul

The ONE View configuration file must contain all variables for executing the application.

```
> cd $SCRATCH/MAQAO_HANDSON/matmul #if cur. directory has changed
> less ov_orig.json
```

```
"executable": "matmul_orig"
"run_command": "<executable> 400 500"
...
"number_processes_per_node": 1
"mpi_command": "srun -p small --reservation=VI-HPS"
...
```

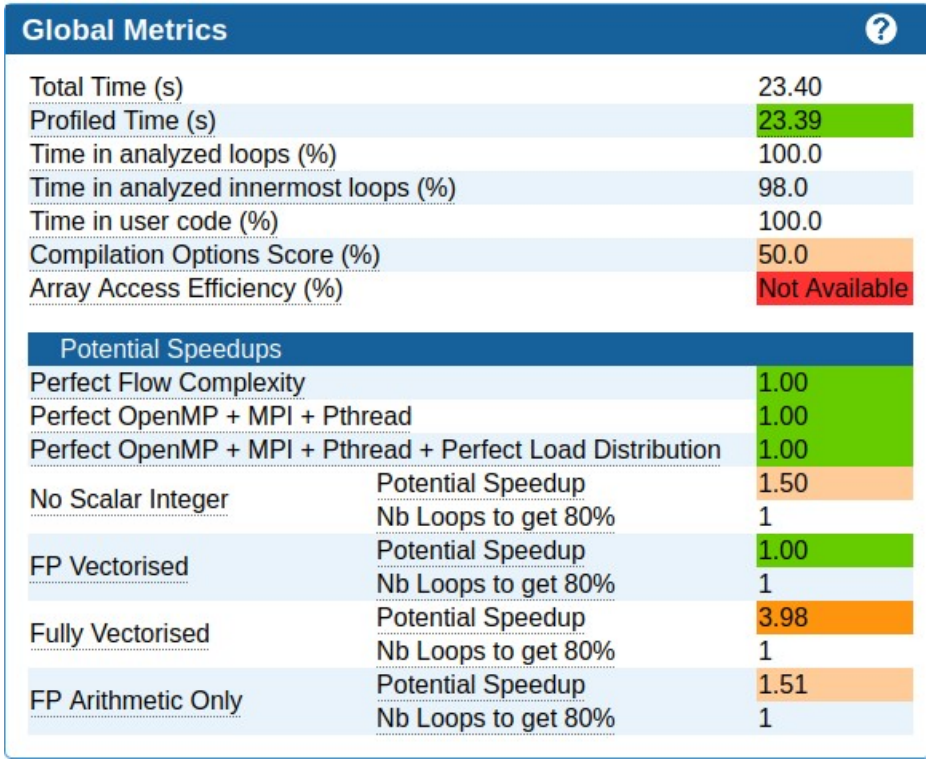
Analyse matrix multiply with ONE View

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

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox turpan_work/MAQAO_HANDSON/matmul/ov_orig/RESULTS/
matmul_one_html/index.html &
```



Global Metrics

Total Time (s)		23.40
Profiled Time (s)		23.39
Time in analyzed loops (%)		100.0
Time in analyzed innermost loops (%)		98.0
Time in user code (%)		100.0
Compilation Options Score (%)		50.0
Array Access Efficiency (%)		Not Available

Potential Speedups

Perfect Flow Complexity		1.00
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.50
	Nb Loops to get 80%	1
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	3.98
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	1.51
	Nb Loops to get 80%	1

CQA output for the baseline kernel

Vectorization

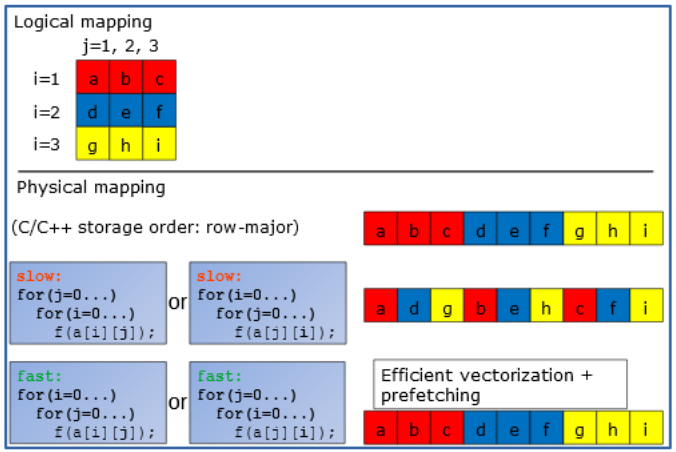
Your loop is not vectorized. 4 data elements could be processed at once in vector registers. By vectorizing your loop, you can lower the cost of an iteration from 3.00 to 0.75 cycles (4.00x speedup).

Details

All VPU 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.

Workaround

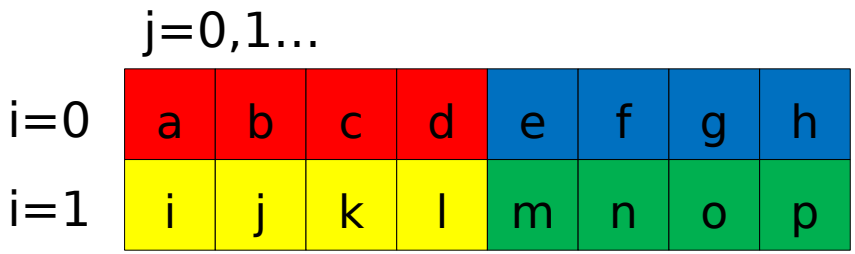
- Try another compiler or update/tune your current one:
  - recompile with fassociative-math (included in Ofast or ffast-math) to extend loop vectorization to FP reductions.
- Remove inter-iterations dependences from your loop and make it unit-stride:
  - If your arrays have 2 or more dimensions, check 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

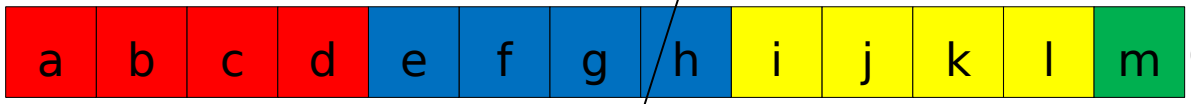
Logical mapping



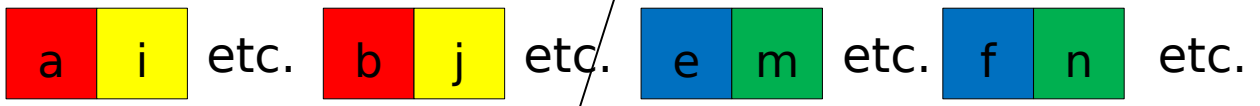
Efficient vectorization + prefetching

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 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

Compile implementation with permuted loops

```
> cd $SCRATCH/MAQAO_HANDSON/matmul #if cur. directory has changed
> make matmul_perm
```

Parameters: <matrix size> <number of repetitions>

```
> srun --reservation=VI-HPS matmul_perm/matmul 400 500
0.17 nanosecond per inner-loop iteration
```

Analyze with maqao

```
> maqao oneview -R1 -xp=ov_perm \
--mpi-command="srun --reservation=VI-HPS" \
-- matmul_perm/matmul 400 500
```

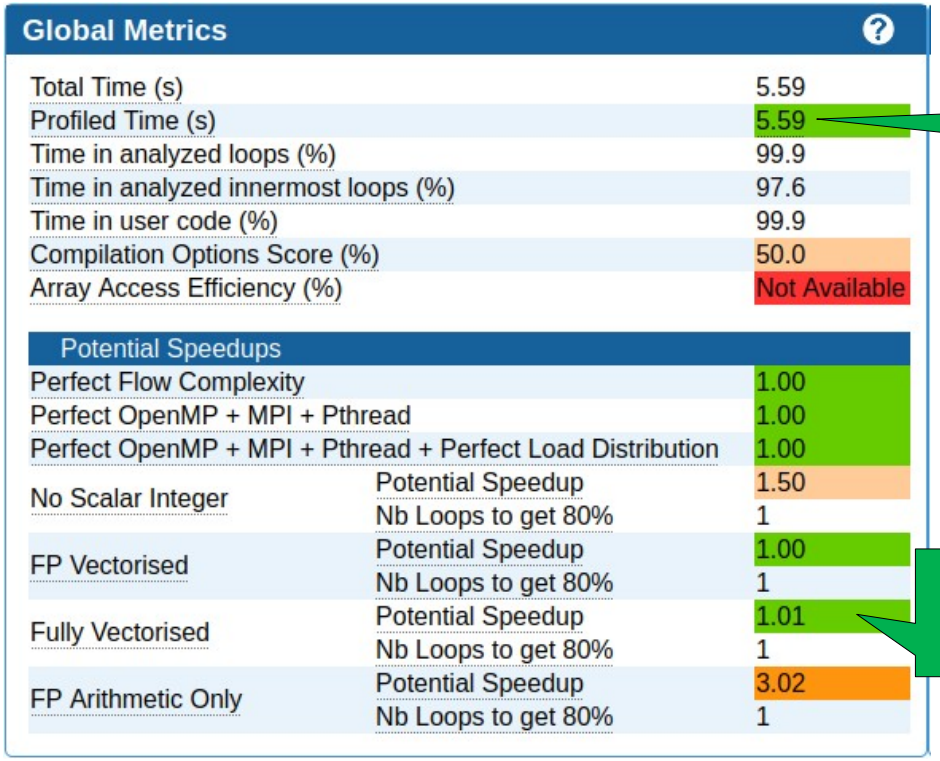
(or use configuration file)

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

Loop permutation results (HTML)

On your local machine (sshfs):

```
> firefox turpan_work/MAQAO_HANDSON/matmul/ov_perm/RESULTS/
matmul_one_html/index.html &
```



Global Metrics

Total Time (s)		5.59
Profiled Time (s)		5.59
Time in analyzed loops (%)		99.9
Time in analyzed innermost loops (%)		97.6
Time in user code (%)		99.9
Compilation Options Score (%)		50.0
Array Access Efficiency (%)		Not Available

Potential Speedups

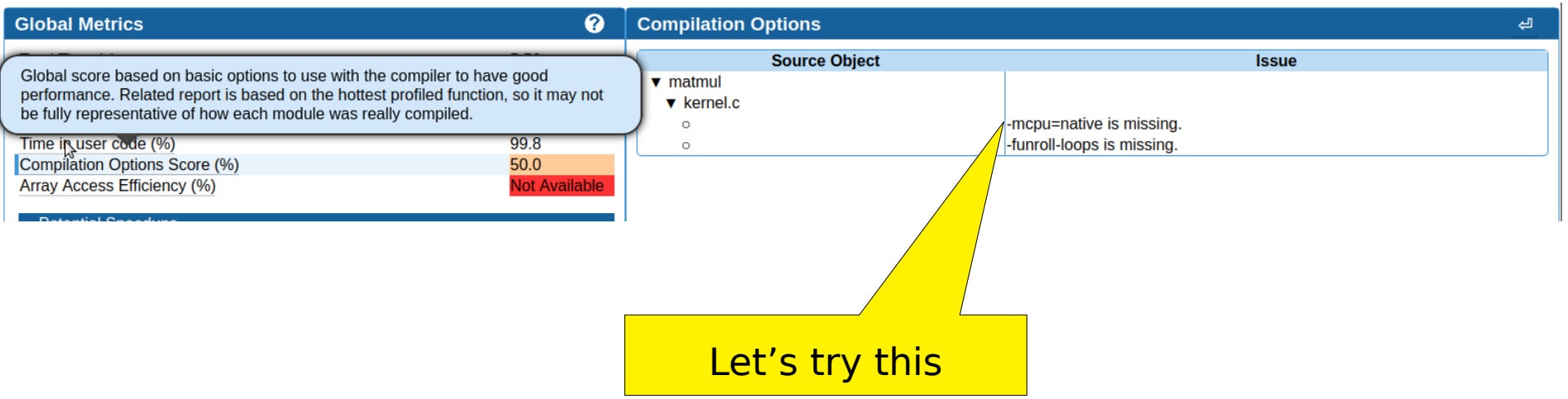
Perfect Flow Complexity		1.00
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.50
	Nb Loops to get 80%	1
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	1.01
	Nb Loops to get 80%	1
FP Arithmetic Only	Potential Speedup	3.02
	Nb Loops to get 80%	1

Faster (was 23.39)

More efficient vectorization (was 3.98)

Next steps

Compilation options



Analyse matrix multiply with permuted loops and compiler optimisation

Compile implementation with permuted loops and compiler optimisation

```
> cd $SCRATCH/MAQAO_HANDSON/matmul #if cur. directory has changed
> make matmul_perm_opt
```

Parameters: <matrix size> <number of repetitions>

```
> srun --reservation=VI-HPS matmul_perm_opt/matmul 400 500
0.13 nanosecond per inner-loop iteration
```

Analyze with maqao

```
> maqao oneview -R1 -xp=ov_perm_opt \
--mpi-command="srun --reservation=VI-HPS" \
-- matmul_perm_opt/matmul 400 500
```

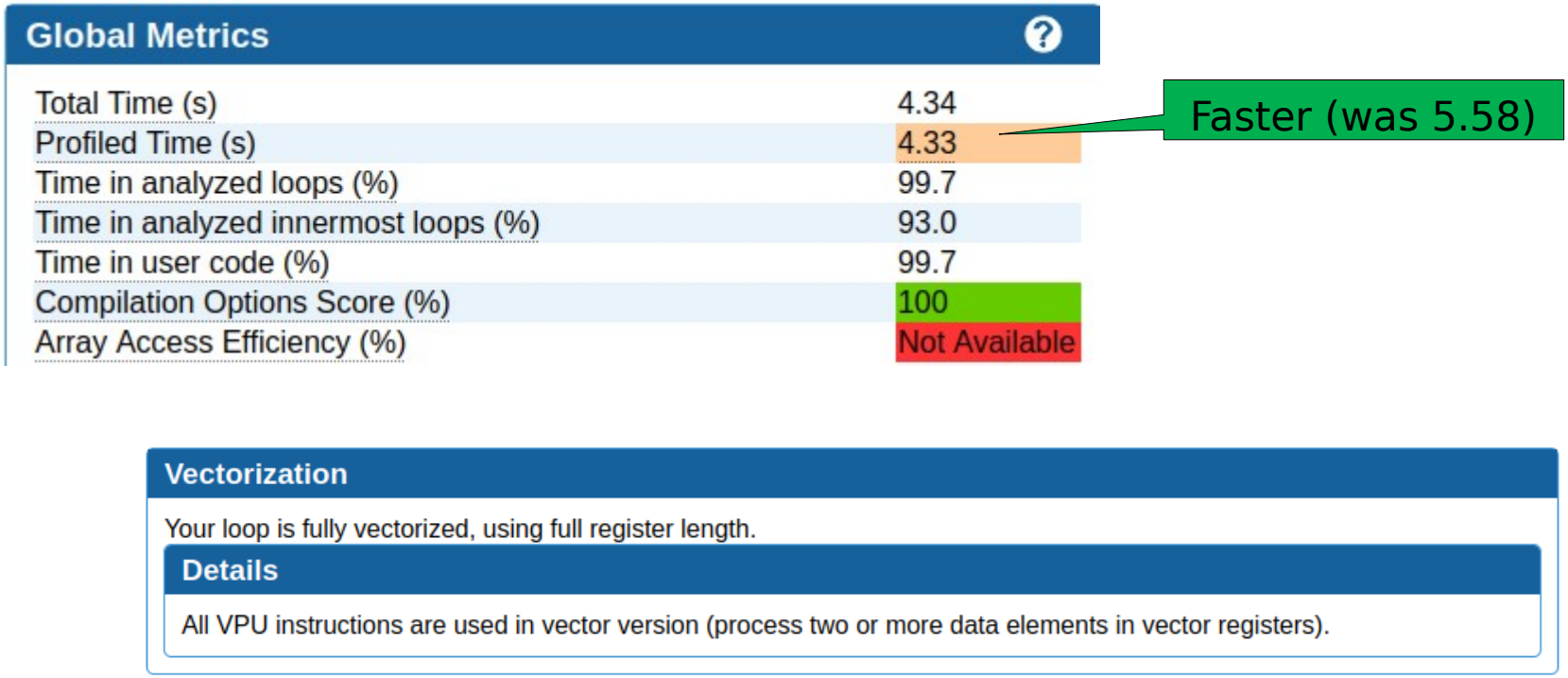
(or use configuration file)

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

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox turpan_work/MAQAO_HANDSON/matmul/ov_perm_opt/RESULTS/
matmul_one_html/index.html &
```



Using comparison mode (iso-source)

```
> maqao oneview --compare-reports \
--inputs=ov_orig,ov_perm,ov_perm_opt -xp=ov_orig_vs_perm
```

Open ov_orig_vs_perm/RESULTS/ov_orig_vs_perm/index.html

▼ **Compared Reports**

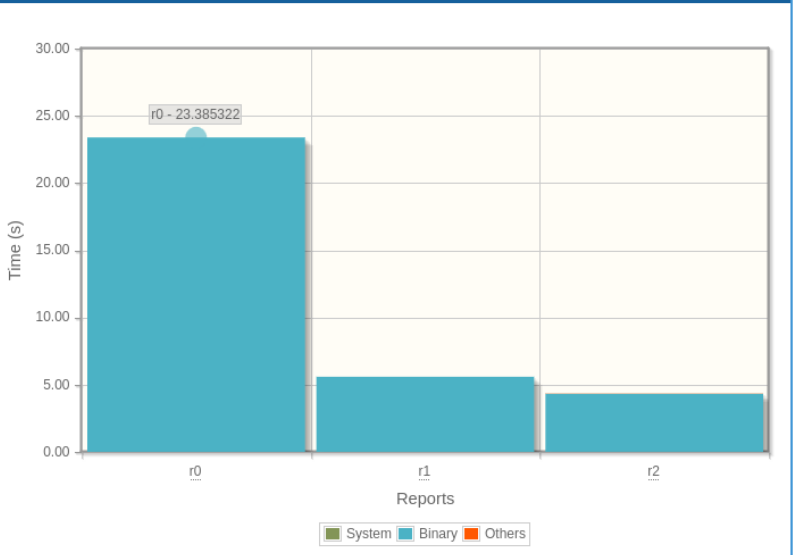
- r0: ov_orig
- r1: ov_perm
- r2: ov_perm_opt

Global Metrics

Metric		r0	r1	r2
Total Time (s)		23.40	5.59	4.34
Profiled Time (s)		23.39	5.59	4.33
Time in analyzed loops (%)		100.0	99.9	99.7
Time in analyzed innermost loops (%)		98.0	97.6	93.0
Time in user code (%)		100.0	99.9	99.7
Compilation Options Score (%)		50.0	50.0	100
Array Access Efficiency (%)		Not Available	Not Available	Not Available
Potential Speedups				
Perfect Flow Complexity		1.00	1.00	1.00
Perfect OpenMP + MPI + Pthread		1.00	1.00	1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00	1.00	1.00
No Scalar Integer	Potential Speedup	1.50	1.50	1.04
	Nb Loops to get 80%	1	1	1
FP Vectorised	Potential Speedup	1.00	1.00	1.00
	Nb Loops to get 80%	1	1	1
Fully Vectorised	Potential Speedup	3.98	1.01	1.01
	Nb Loops to get 80%	1	1	1
Only FP Arithmetic	Potential Speedup	1.51	3.02	3.10
	Nb Loops to get 80%	1	1	2

Application Categorization

Time



Comparison: loop view

Comparison performed based on loop source information

MAQAO Global Functions Loops

Show All Loops | Order by Coverages | Order by Locations

Loops

▼ kernel.c: 24 - 288.57%

Run ov_orig
Loop Source Regions: • /tmpdir/oseret/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	22.93	22.93	98.03	0	25

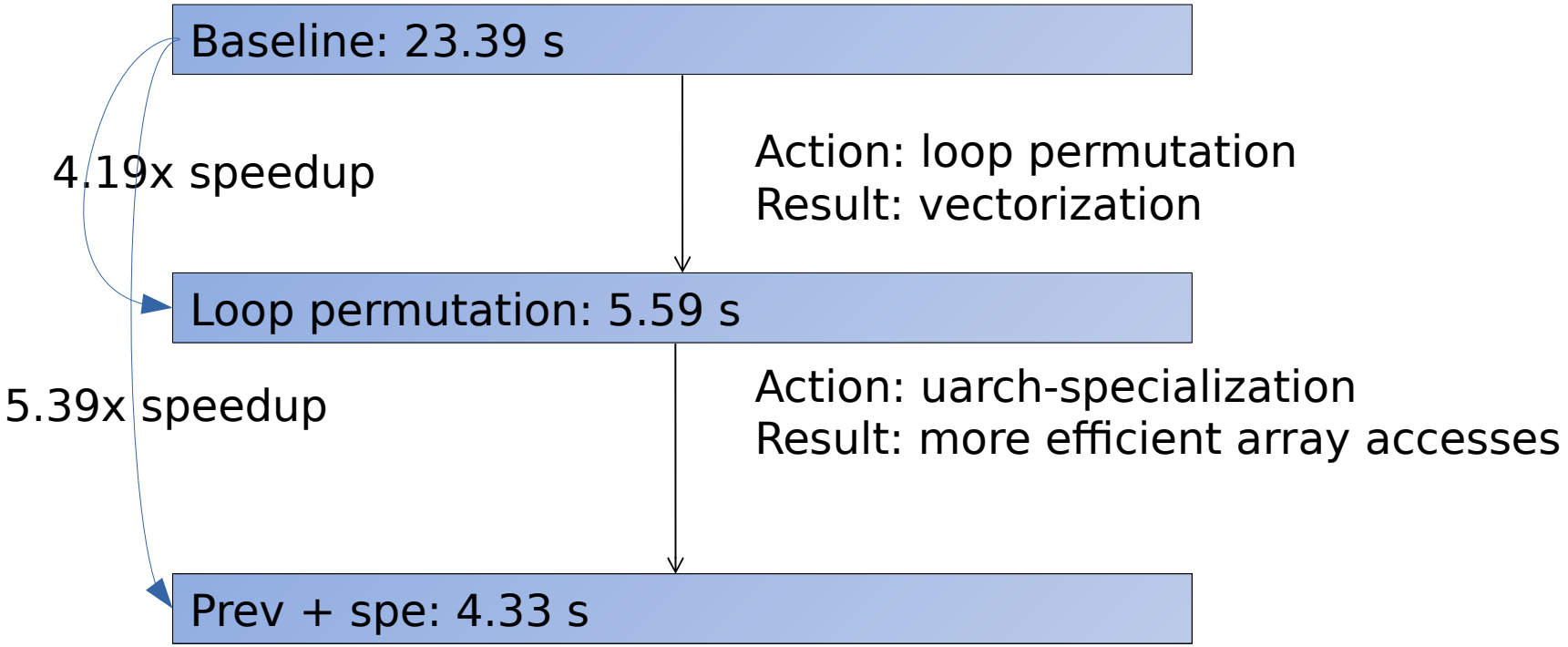
Run ov_perm
Loop Source Regions: • /tmpdir/oseret/MAQAO_HANDSON/matmul/matmul_perm/kernel.c: 24-25

Assembly Loop ID	Max Time Over Threads (s)	Time w.r.t. Wall Time (s)	Cov (%)	Vect. Ratio (%)	Vector Length Use (%)
5	5.45	5.45	97.58	100	100

Run ov_perm_opt
Loop Source Regions: • /tmpdir/oseret/MAQAO_HANDSON/matmul/matmul_perm/kernel.c: 24-25

Assembly Loop ID	Max Time Over Threads (s)	Time w.r.t. Wall Time (s)	Cov (%)	Vect. Ratio (%)	Vector Length Use (%)
3	4.03	4.03	92.96	100	100

Summary of optimizations and gains



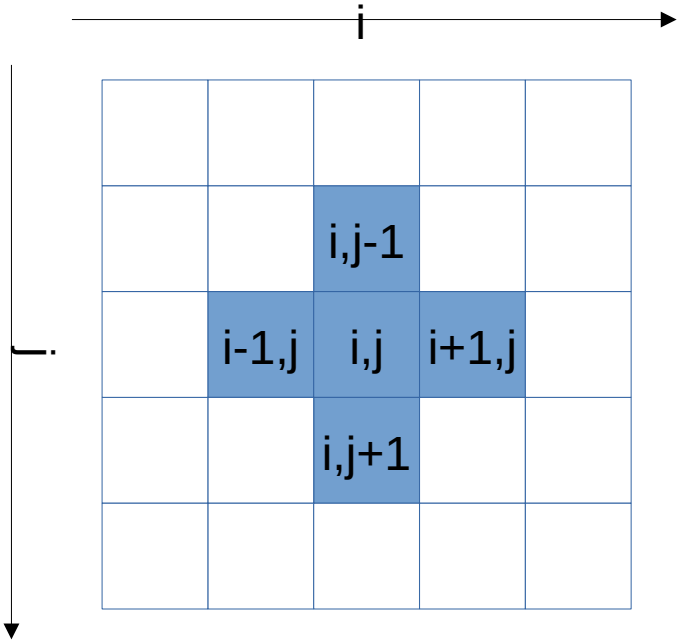
Hydro code

```
int build_index (int i, int j, int grid_size)
{
  return (i + (grid_size + 2) * j);
}

void linearSolver0 (...) {
  int i, j, k;

  for (k=0; k<20; k++)
    for (i=1; i<=grid_size; i++)
      for (j=1; j<=grid_size; j++)
        x[build_index(i, j, grid_size)] =
  (a * ( x[build_index(i-1, j, grid_size)] +
         x[build_index(i+1, j, grid_size)] +
         x[build_index(i, j-1, grid_size)] +
         x[build_index(i, j+1, grid_size)]
       ) + x0[build_index(i, j, grid_size)]
  ) / c;
}
```

Iterative linear system solver using the Gauss-Siedel relaxation technique.
« Stencil » code



Compile with ARM compiler on login node

Switch to the hydro handson folder

```
> cd $SCRATCH/MAQAO_HANDSON/hydro
```

Load MAQAO (if necessary)

```
> module use /tmpdir/vi-hps/opt/modules
> module load maqao
```

Load compiler environment (if necessary)

```
> module load arm acfl/22.1
```

Compile

```
> make
```

Running original kernel

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

```
> srun --reservation=VI-HPS ./hydro_orig 300 200
Nanoseconds per element for solvers: 783.92
```

Profiling with MAQAO

The ONE View configuration file must contain all variables for executing the application.

```
> cd $SCRATCH/MAQAO_HANDSON/hydro #if cur. directory has changed
> less ov_orig.json
```

```
"executable": "./hydro_orig"
"run_command": "<executable> 300 200"
...
number_processes_per_node = 1
mpi_command = "srun -p small --reservation=VI-HPS"
...
```

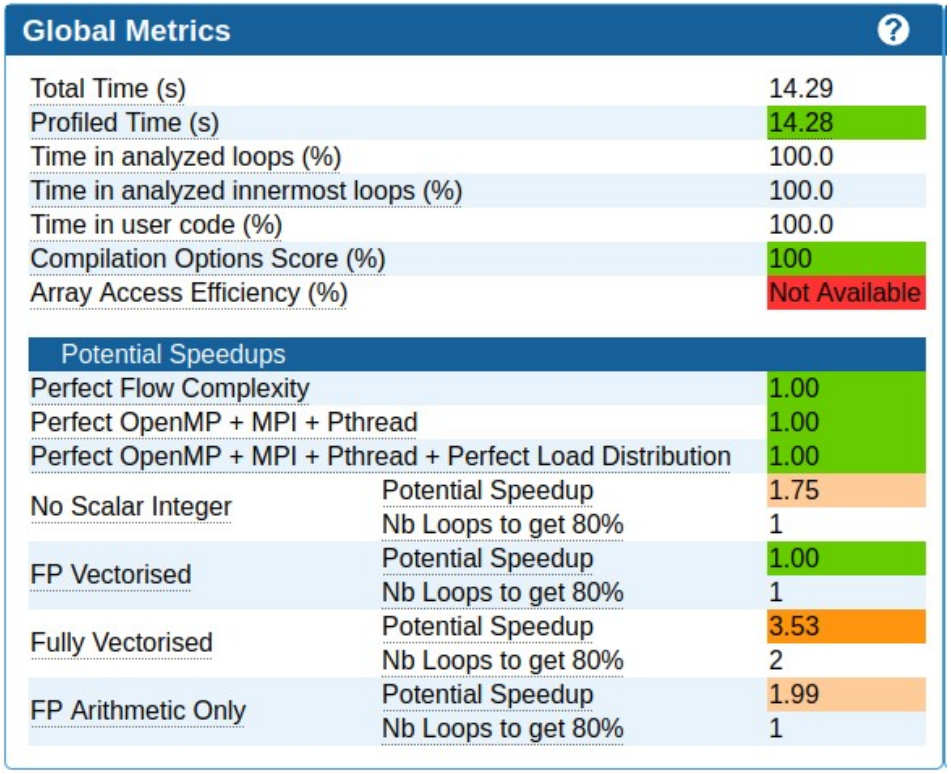
Profile with MAQAO

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

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox turpan_work/MAQAO_HANDSON/hydro/ov_orig/RESULTS/
hydro_orig_one_html/index.html &
```



Global Metrics

Total Time (s)	14.29
Profiled Time (s)	14.28
Time in analyzed loops (%)	100.0
Time in analyzed innermost loops (%)	100.0
Time in user code (%)	100.0
Compilation Options Score (%)	100
Array Access Efficiency (%)	Not Available

Potential Speedups

Perfect Flow Complexity		1.00
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.75
	Nb Loops to get 80%	1
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	3.53
	Nb Loops to get 80%	2
FP Arithmetic Only	Potential Speedup	1.99
	Nb Loops to get 80%	1

Running and analyzing the original kernel

Source Code

/users/m23012/m23012vc/PREP_HANDSON/SCRATCH/MAQAO_HANDSON/hydro/kernel.c: 91 - 97

```
91:          for (j = 1; j <= grid_size; j++) {
92:             x[build_index(i, j, grid_size)] =
93:               (a * ( x[build_index(i-1, j, grid_size)] +
94:                      x[build_index(i+1, j, grid_size)] +
95:                      x[build_index(i, j-1, grid_size)] +
96:                      x[build_index(i, j+1, grid_size)]) +
97:                x0[build_index(i, j, grid_size)]) / c;
```

CQA

gain | potential | hint | expert

Code clean check

Detected a slowdown caused by scalar integer instructions (typically used for address computation). By removing them, you can lower the cost of an iteration from 6.00 to 3.50 cycles (1.71x speedup).

Workaround

- Try to reorganize arrays of structures to structures of arrays
- Consider to permute loops (see vectorization gain report)

Vectorization

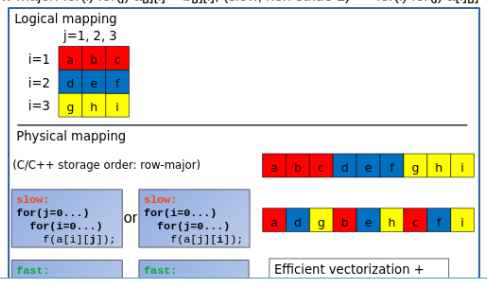
Your loop is not vectorized. Only 31% of vector register length is used (average across all VPU instructions). By vectorizing your loop, you can lower the cost of an iteration from 6.00 to 1.67 cycles (3.60x speedup).

Details

All VPU 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.

Workaround

- Try another compiler or update/tune your current one:
  - recompile with ffast-math (included in Ofast) 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)



Profiling with MAQAO after loop permutation

The ONE View configuration file must contain all variables for executing the application.

```
> cd $SCRATCH/MAQAO_HANDSON/hydro #if cur. directory has changed
> less ov_perm.json
```

```
"executable": "./hydro_perm"
"run_command": "<executable> 300 200"
...
number_processes_per_node = 1
mpi_command = "srun -p small --reservation=VI-HPS"
...
```

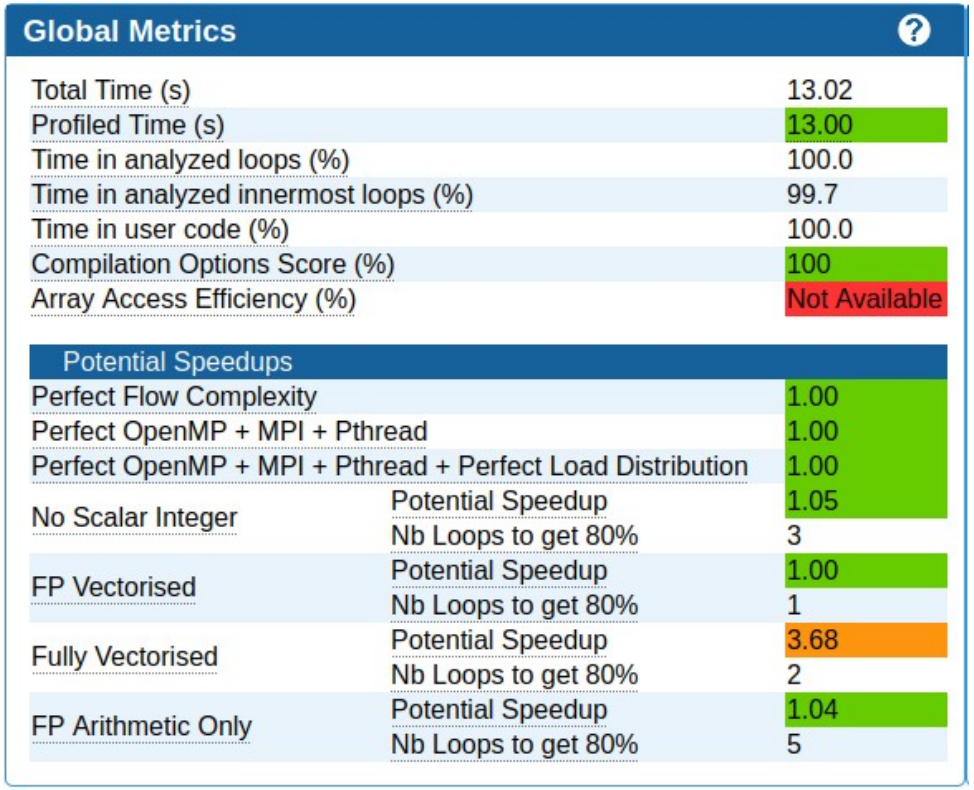
Profile with MAQAO

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

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox turpan_work/MAQAO_HANDSON/hydro/ov_perm/RESULTS/
hydro_perm_one_html/index.html &
```



Global Metrics

Total Time (s)	13.02
Profiled Time (s)	13.00
Time in analyzed loops (%)	100.0
Time in analyzed innermost loops (%)	99.7
Time in user code (%)	100.0
Compilation Options Score (%)	100
Array Access Efficiency (%)	Not Available

Potential Speedups

Perfect Flow Complexity		1.00
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.05
	Nb Loops to get 80%	3
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	3.68
	Nb Loops to get 80%	2
FP Arithmetic Only	Potential Speedup	1.04
	Nb Loops to get 80%	5

CQA output for orig/perm

The related source loop is not unrolled or unrolled with no peel/tail loop.

gain | potential | **hint** | expert

Slow data structures access

Detected data structures (typically arrays) that cannot be efficiently read/written

Details

- Constant unknown stride: 4 occurrence(s)

Non-unit stride (uncontiguous) accesses are not efficiently using data caches

Workaround

- Try to reorganize arrays of structures to structures of arrays
- Consider to permute loops (see vectorization gain report)

Type of elements and instruction set

5 SSE or AVX instructions are processing arithmetic or math operations on single precision FP elements in scalar mode (one at a time).

Matching between your loop (in the source code) and the binary loop

The binary loop is composed of 5 FP arithmetical operations:

- 4: addition or subtraction
- 1: multiply

The binary loop is loading 20 bytes (5 single precision FP elements). The binary loop is storing 4 bytes (1 single precision FP elements).

Arithmetic intensity

Arithmetic intensity is 0.21 FP operations per loaded or stored byte.

Unroll opportunity

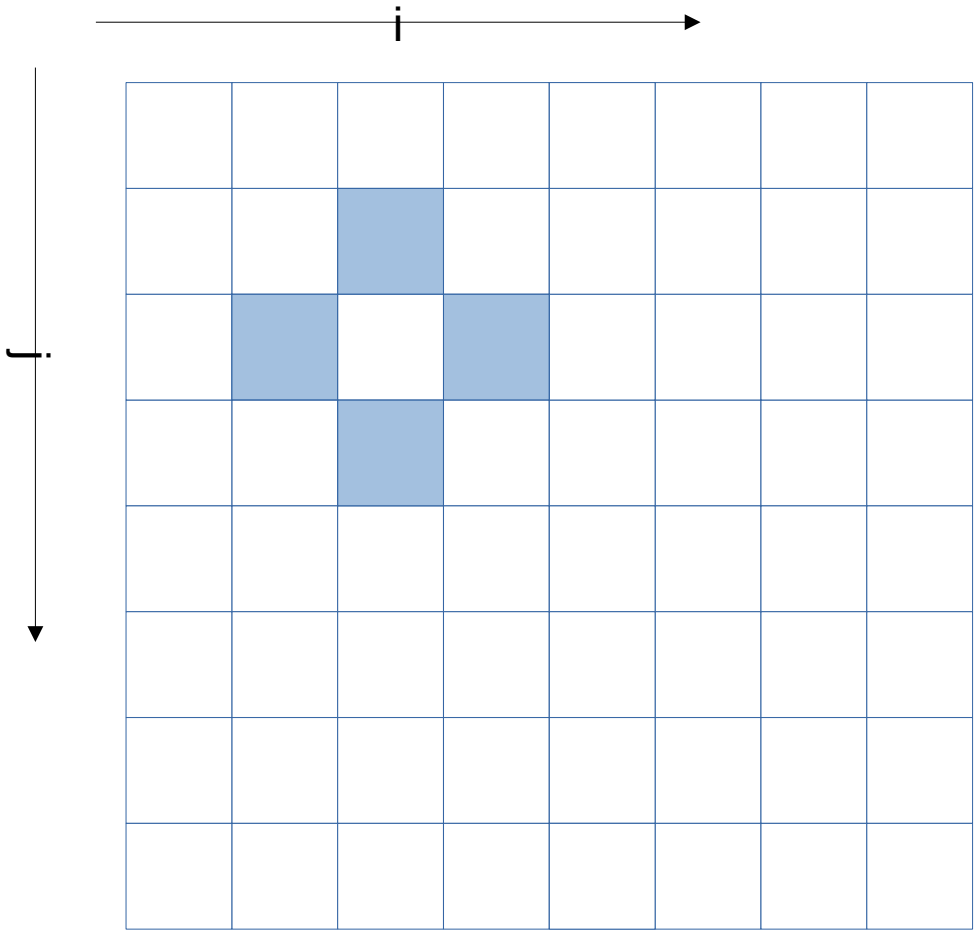
Loop is potentially data access bound.

Workaround

Unroll your loop if trip count is significantly higher than target unroll factor and if some data references are common to consecutive iterations. This can be done manually. Or by combining O2/O3 with the UNROLL (resp. UNROLL_AND_JAM) directive on top of the inner (resp. surrounding) loop. You can enforce an unroll factor: e.g. UNROLL(4).

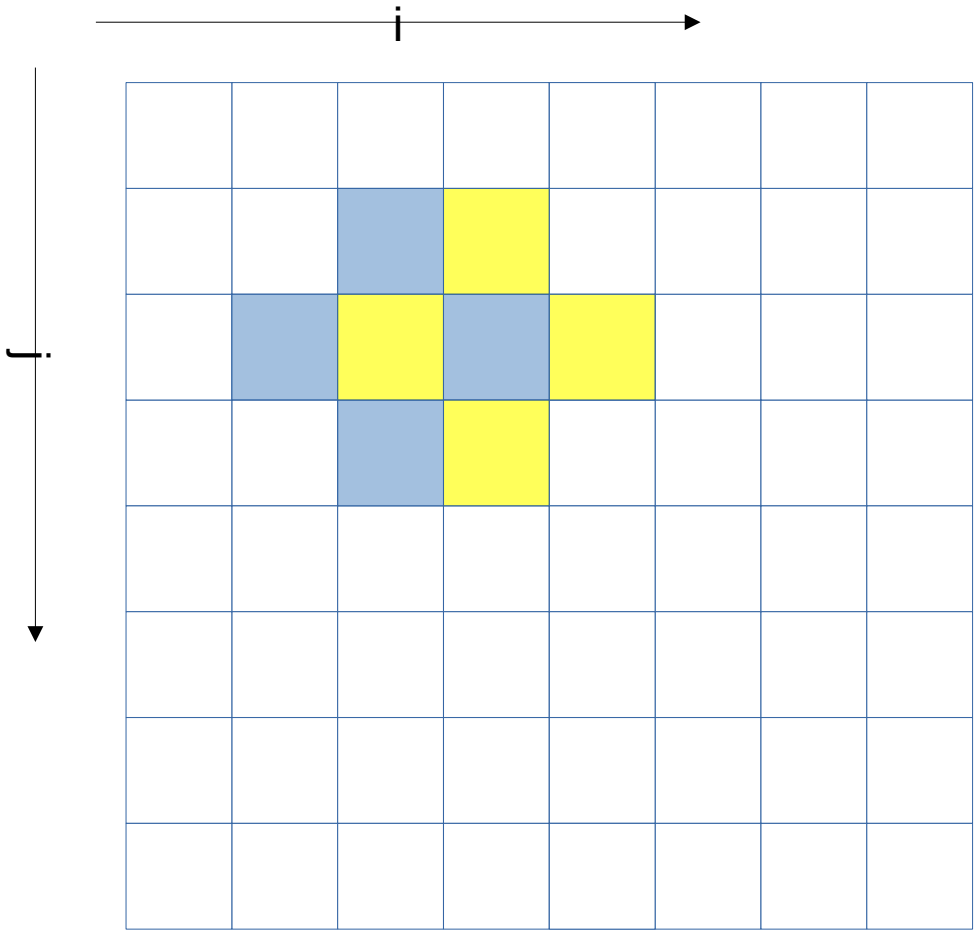
Unrolling is generally a good deal: fast to apply and often provides gain. Let's try to reuse data references through unrolling

Memory references reuse : 4x4 unroll footprint on loads



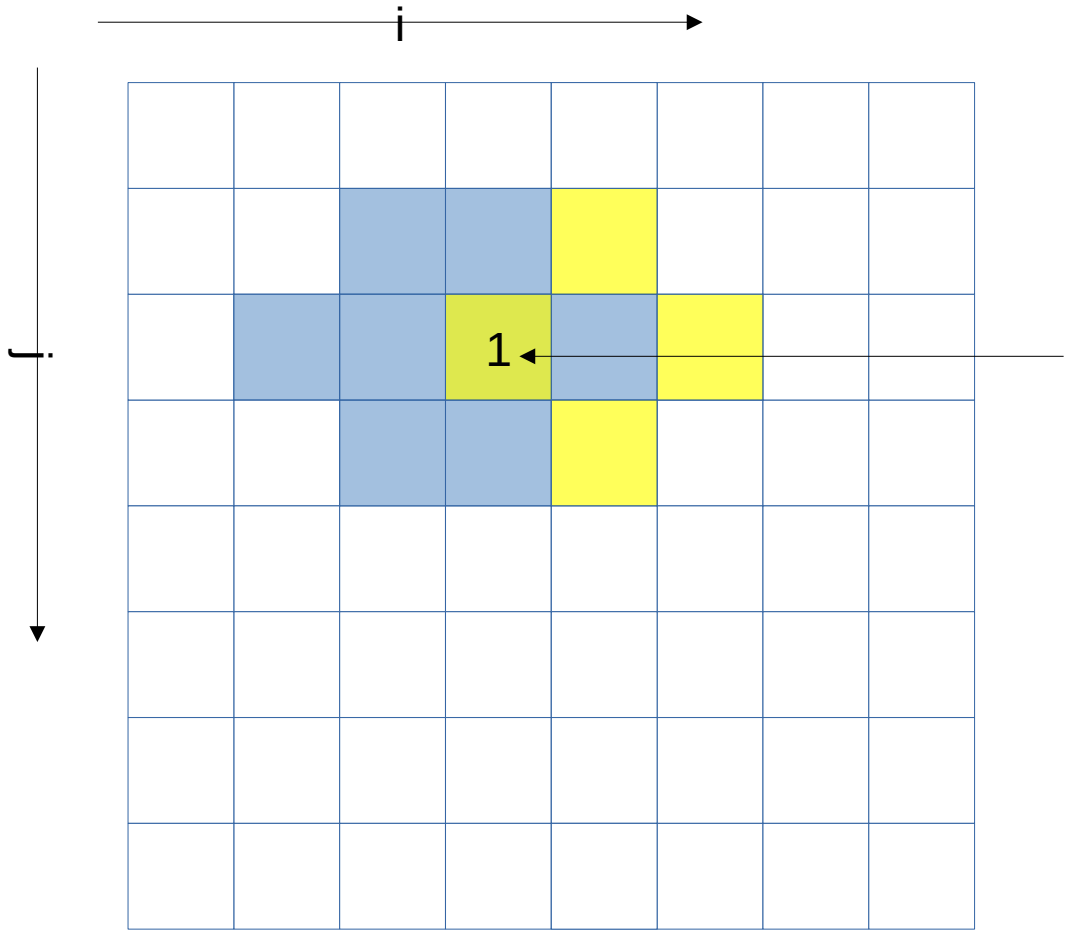
LINEAR_SOLVER(i+0,j+0)

Memory references reuse : 4x4 unroll footprint on loads



LINEAR_SOLVER(i+0,j+0)
LINEAR_SOLVER(i+1,j+0)

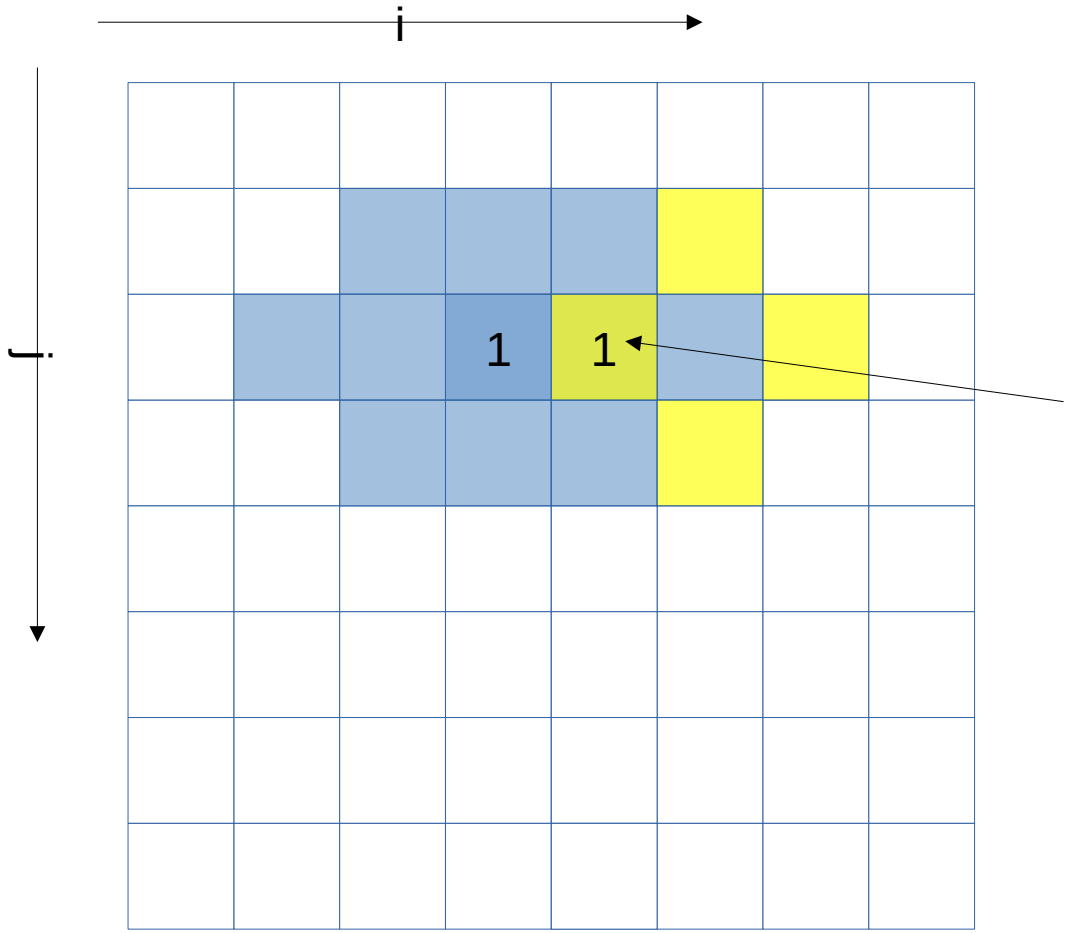
Memory references reuse : 4x4 unroll footprint on loads



LINEAR_SOLVER(i+0,j+0)
LINEAR_SOLVER(i+1,j+0)
LINEAR_SOLVER(i+2,j+0)

1 reuse

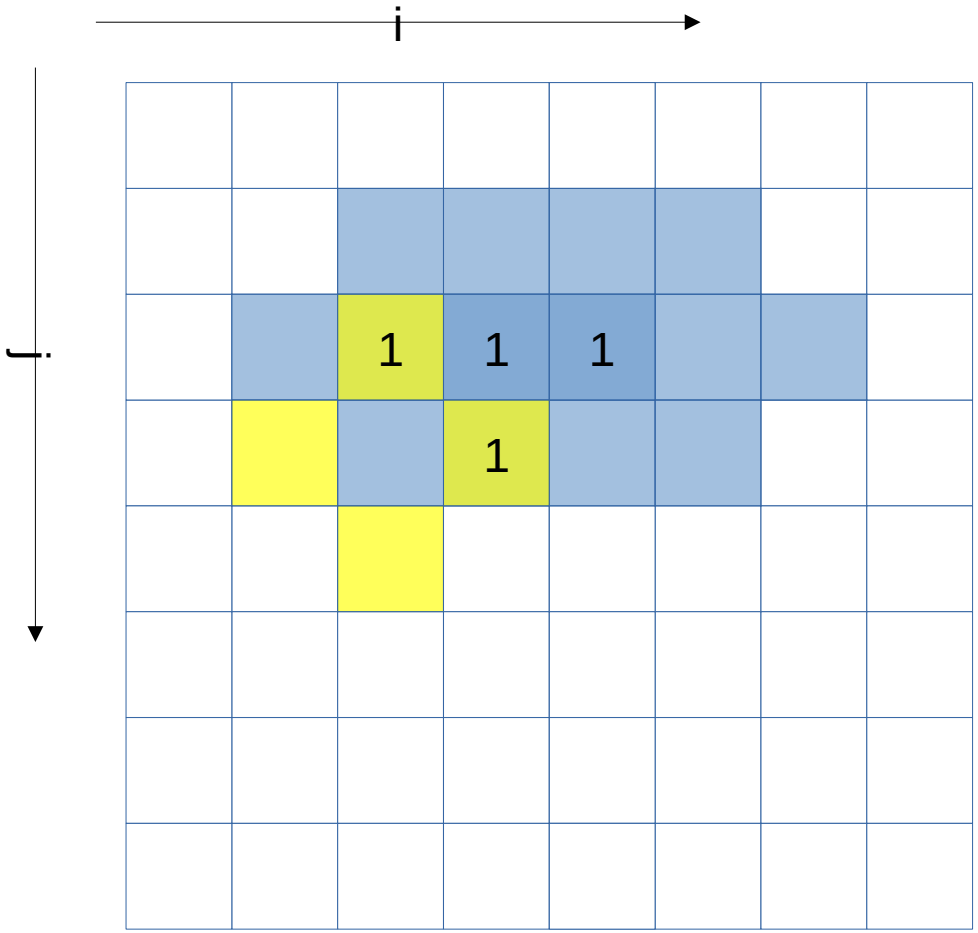
Memory references reuse : 4x4 unroll footprint on loads



LINEAR_SOLVER(i+0,j+0)
LINEAR_SOLVER(i+1,j+0)
LINEAR_SOLVER(i+2,j+0)
LINEAR_SOLVER(i+3,j+0)

2 reuses

Memory references reuse : 4x4 unroll footprint on loads

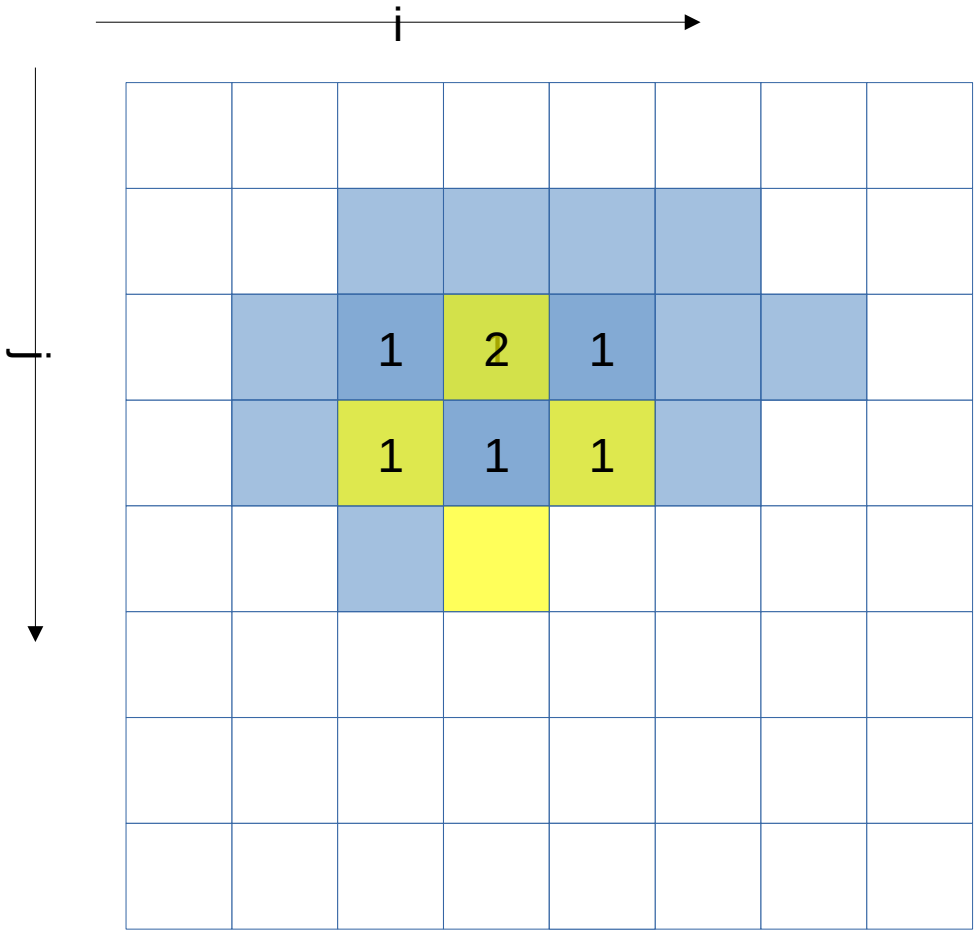


LINEAR_SOLVER(i+0,j+0)
LINEAR_SOLVER(i+1,j+0)
LINEAR_SOLVER(i+2,j+0)
LINEAR_SOLVER(i+3,j+0)

LINEAR_SOLVER(i+0,j+1)

4 reuses

Memory references reuse : 4x4 unroll footprint on loads

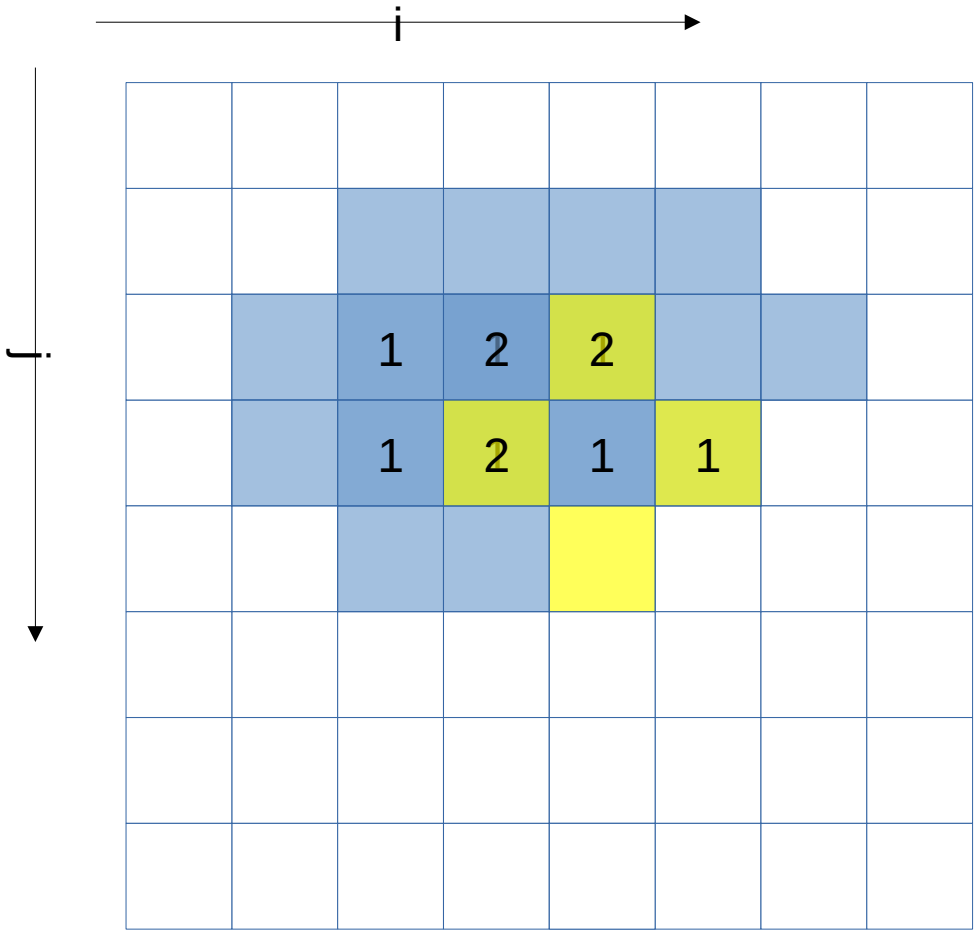


LINEAR_SOLVER(i+0,j+0)
LINEAR_SOLVER(i+1,j+0)
LINEAR_SOLVER(i+2,j+0)
LINEAR_SOLVER(i+3,j+0)

LINEAR_SOLVER(i+0,j+1)
LINEAR_SOLVER(i+1,j+1)

7 reuses

Memory references reuse : 4x4 unroll footprint on loads

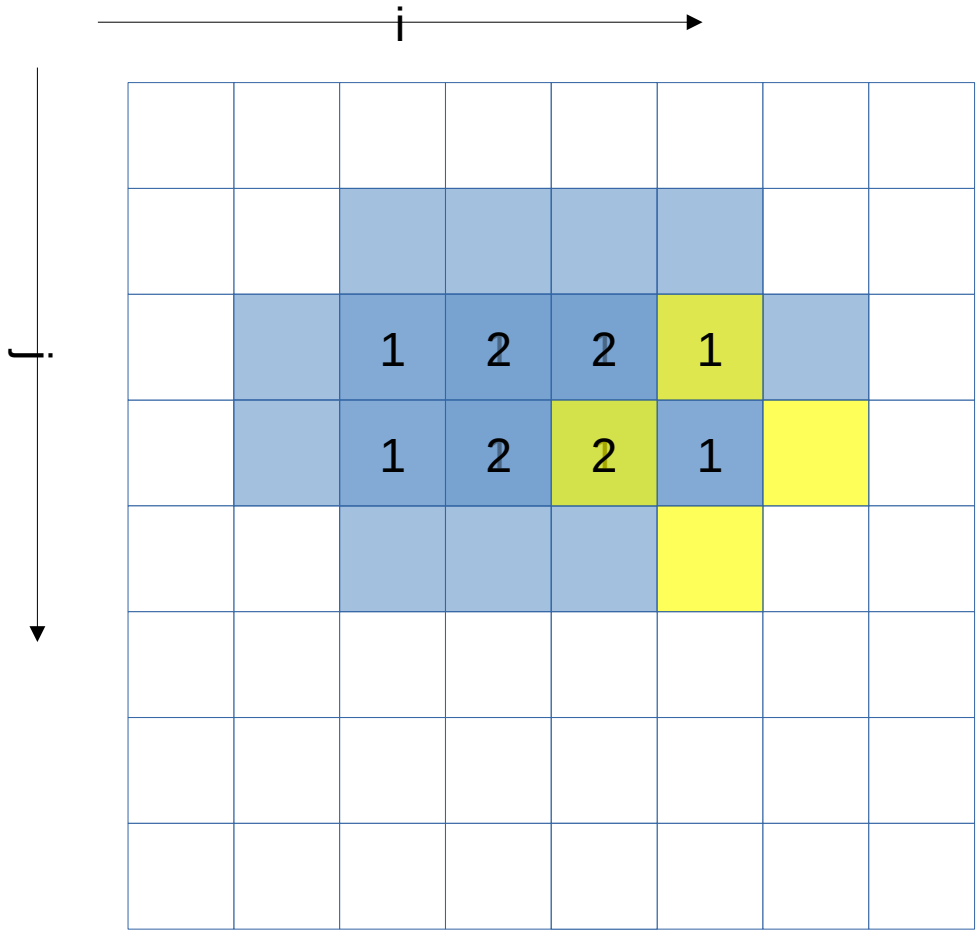


LINEAR_SOLVER(i+0,j+0)
LINEAR_SOLVER(i+1,j+0)
LINEAR_SOLVER(i+2,j+0)
LINEAR_SOLVER(i+3,j+0)

LINEAR_SOLVER(i+0,j+1)
LINEAR_SOLVER(i+1,j+1)
LINEAR_SOLVER(i+2,j+1)

10 reuses

Memory references reuse : 4x4 unroll footprint on loads

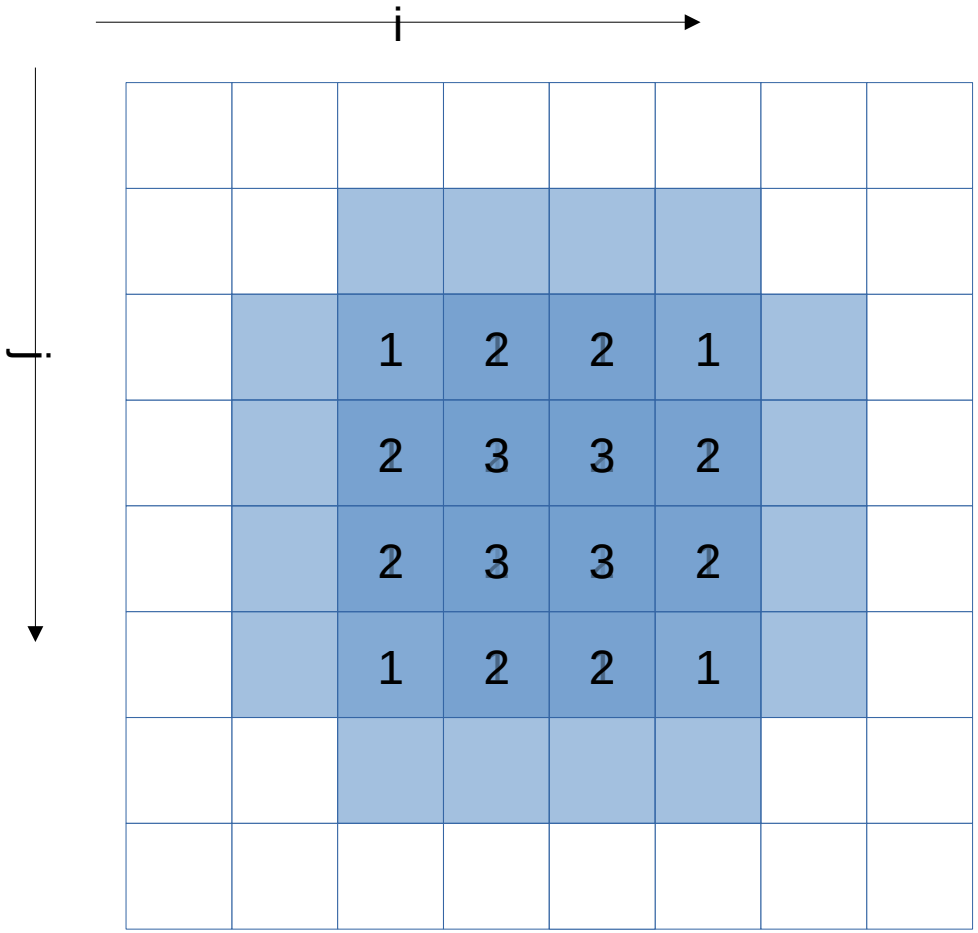


LINEAR_SOLVER(i+0,j+0)
LINEAR_SOLVER(i+1,j+0)
LINEAR_SOLVER(i+2,j+0)
LINEAR_SOLVER(i+3,j+0)

LINEAR_SOLVER(i+0,j+1)
LINEAR_SOLVER(i+1,j+1)
LINEAR_SOLVER(i+2,j+1)
LINEAR_SOLVER(i+3,j+1)

12 reuses

Memory references reuse : 4x4 unroll footprint on loads



LINEAR_SOLVER(i+0-3,j+0)

LINEAR_SOLVER(i+0-3,j+1)

LINEAR_SOLVER(i+0-3,j+2)

LINEAR_SOLVER(i+0-3,j+3)

32 reuses

Impacts of memory reuse

- For the x array, instead of 4x4x4 = 64 loads, now only 32 (32 loads avoided by reuse)
- For the x0 array no reuse possible : 16 loads
- Total loads : 48 instead of 80

4x4 unroll

```
#define LINEARSOLVER(...) x[build_index(i, j, grid_size)] = …

void linearSolver2 (...) {
  (...)

  for (k=0; k<20; k++)
    for (i=1; i<=grid_size-3; i+=4)
      for (j=1; j<=grid_size-3; j+=4) {
        LINEARSOLVER (…, i+0, j+0);
        LINEARSOLVER (…, i+0, j+1);
        LINEARSOLVER (…, i+0, j+2);
        LINEARSOLVER (…, i+0, j+3);

        LINEARSOLVER (…, i+1, j+0);
        LINEARSOLVER (…, i+1, j+1);
        LINEARSOLVER (…, i+1, j+2);
        LINEARSOLVER (…, i+1, j+3);

        LINEARSOLVER (…, i+2, j+0);
        LINEARSOLVER (…, i+2, j+1);
        LINEARSOLVER (…, i+2, j+2);
        LINEARSOLVER (…, i+2, j+3);

        LINEARSOLVER (…, i+3, j+0);
        LINEARSOLVER (…, i+3, j+1);
        LINEARSOLVER (…, i+3, j+2);
        LINEARSOLVER (…, i+3, j+3);
      }
}
```

grid_size must now be multiple of 4. Or loop control must be adapted (much less readable) to handle leftover iterations

Profiling with MAQAO the 4x4 unrolled kernel

The ONE View configuration file must contain all variables for executing the application.

```
> cd $SCRATCH/MAQAO_HANDSON/hydro #if cur. directory has changed
> less ov_unroll.json
```

```
"executable": "./hydro_unroll"
"run_command": "<executable> 300 200"
...
number_processes_per_node = 1
mpi_command = "srun -p small --reservation=VI-HPS"
...
```

Profile with MAQAO

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

Viewing results (HTML)

On your local machine (sshfs):

```
> firefox turpan_work/MAQAO_HANDSON/hydro/ov_unroll/RESULTS/
hydro_unroll_one_html/index.html &
```

Global Metrics

Total Time (s)	6.00
Profiled Time (s)	5.99
Time in analyzed loops (%)	99.8
Time in analyzed innermost loops (%)	99.8
Time in user code (%)	99.8
Compilation Options Score (%)	100
Array Access Efficiency (%)	Not Available

Potential Speedups

Perfect Flow Complexity		1.00
Perfect OpenMP + MPI + Pthread		1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00
No Scalar Integer	Potential Speedup	1.11
	Nb Loops to get 80%	2
FP Vectorised	Potential Speedup	1.00
	Nb Loops to get 80%	1
Fully Vectorised	Potential Speedup	3.72
	Nb Loops to get 80%	3
FP Arithmetic Only	Potential Speedup	1.25
	Nb Loops to get 80%	3

Analyzing 4x4 unrolled kernel

Source Code

/tmpdir/oseret/MAQAO_HANDSON/hydro/kernel.c: 139 - 158

```
139:          for (i = 1; i <= grid_size-3; i+=4) {
140:             LINEARSOLVER (x, x0, a, c, grid_size, i+0, j+0);
141:             LINEARSOLVER (x, x0, a, c, grid_size, i+1, j+0);
142:             LINEARSOLVER (x, x0, a, c, grid_size, i+2, j+0);
143:             LINEARSOLVER (x, x0, a, c, grid_size, i+3, j+0);
144:
145:             LINEARSOLVER (x, x0, a, c, grid_size, i+0, j+1);
146:             LINEARSOLVER (x, x0, a, c, grid_size, i+1, j+1);
147:             LINEARSOLVER (x, x0, a, c, grid_size, i+2, j+1);
148:             LINEARSOLVER (x, x0, a, c, grid_size, i+3, j+1);
149:
150:             LINEARSOLVER (x, x0, a, c, grid_size, i+0, j+2);
151:             LINEARSOLVER (x, x0, a, c, grid_size, i+1, j+2);
152:             LINEARSOLVER (x, x0, a, c, grid_size, i+2, j+2);
153:             LINEARSOLVER (x, x0, a, c, grid_size, i+3, j+2);
154:
155:             LINEARSOLVER (x, x0, a, c, grid_size, i+0, j+3);
156:             LINEARSOLVER (x, x0, a, c, grid_size, i+1, j+3);
157:             LINEARSOLVER (x, x0, a, c, grid_size, i+2, j+3);
158:             LINEARSOLVER (x, x0, a, c, grid_size, i+3, j+3);
```

CQA

The related source loop is not unrolled or unrolled with no peel/tail loop.

gain | potential | hint | expert

Vectorization

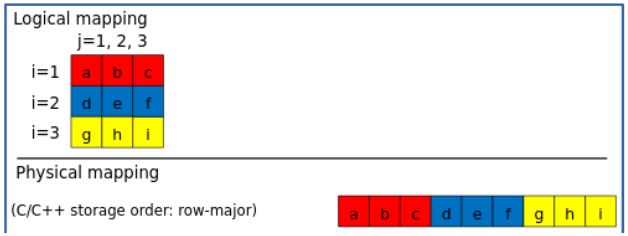
Your loop is not vectorized. Only 25% of vector register length is used (average across all VPU instructions). By vectorizing your loop, you can lower the cost of an iteration from 56.00 to 14.00 cycles (4.00x speedup).

Details

All VPU 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.

Workaround

- Try another compiler or update/tune your current one:
  - recompile with ffast-math (included in Ofast) 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)



Analyzing 4x4 unrolled kernel

Matching between your loop (in the source code) and the binary loop

The binary loop is composed of 96 FP arithmetical operations:

- 64: addition or subtraction
- 16: multiply
- 16: divide

The binary loop is loading 260 bytes. The binary loop is storing 64 bytes.

4x4 Unrolling were applied

corresponds to 65 FP32 elements (better than 80 even if worse than 48 expected)

Using comparison mode (iso-source)

```
> maqao oneview --compare-reports \
--inputs=ov_orig,ov_perm,ov_unroll -xp=ov_orig_vs_unroll
```

Open ov_orig_vs_unroll/RESULTS/ov_orig_vs_unroll/index.html

▼ **Compared Reports**

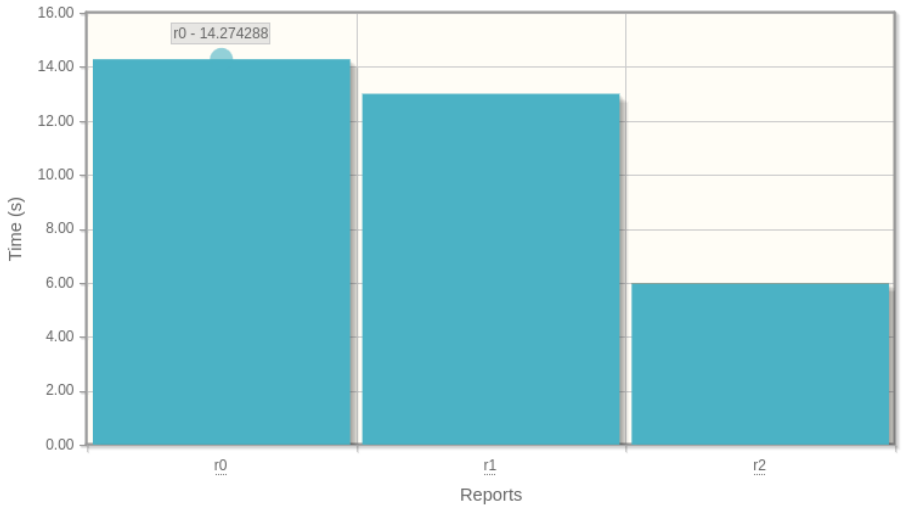
- r0: ov_orig
- r1: ov_perm
- r2: ov_unroll

Global Metrics

Metric		r0	r1	r2
Total Time (s)		14.29	13.02	6.00
Profiled Time (s)		14.28	13.00	5.99
Time in analyzed loops (%)		100.0	100.0	99.8
Time in analyzed innermost loops (%)		100.0	99.7	99.8
Time in user code (%)		100.0	100.0	99.8
Compilation Options Score (%)		100	100	100
Array Access Efficiency (%)		Not Available	Not Available	Not Available
Potential Speedups				
Perfect Flow Complexity		1.00	1.00	1.00
Perfect OpenMP + MPI + Pthread		1.00	1.00	1.00
Perfect OpenMP + MPI + Pthread + Perfect Load Distribution		1.00	1.00	1.00
No Scalar Integer	Potential Speedup	1.75	1.05	1.11
	Nb Loops to get 80%	1	3	2
FP Vectorised	Potential Speedup	1.00	1.00	1.00
	Nb Loops to get 80%	1	1	1
Fully Vectorised	Potential Speedup	3.53	3.68	3.72
	Nb Loops to get 80%	2	2	3
Only FP Arithmetic	Potential Speedup	1.99	1.04	1.25
	Nb Loops to get 80%	1	5	3

Application Categorization

Time



Comparison: function view

Comparison performed based on loop source information

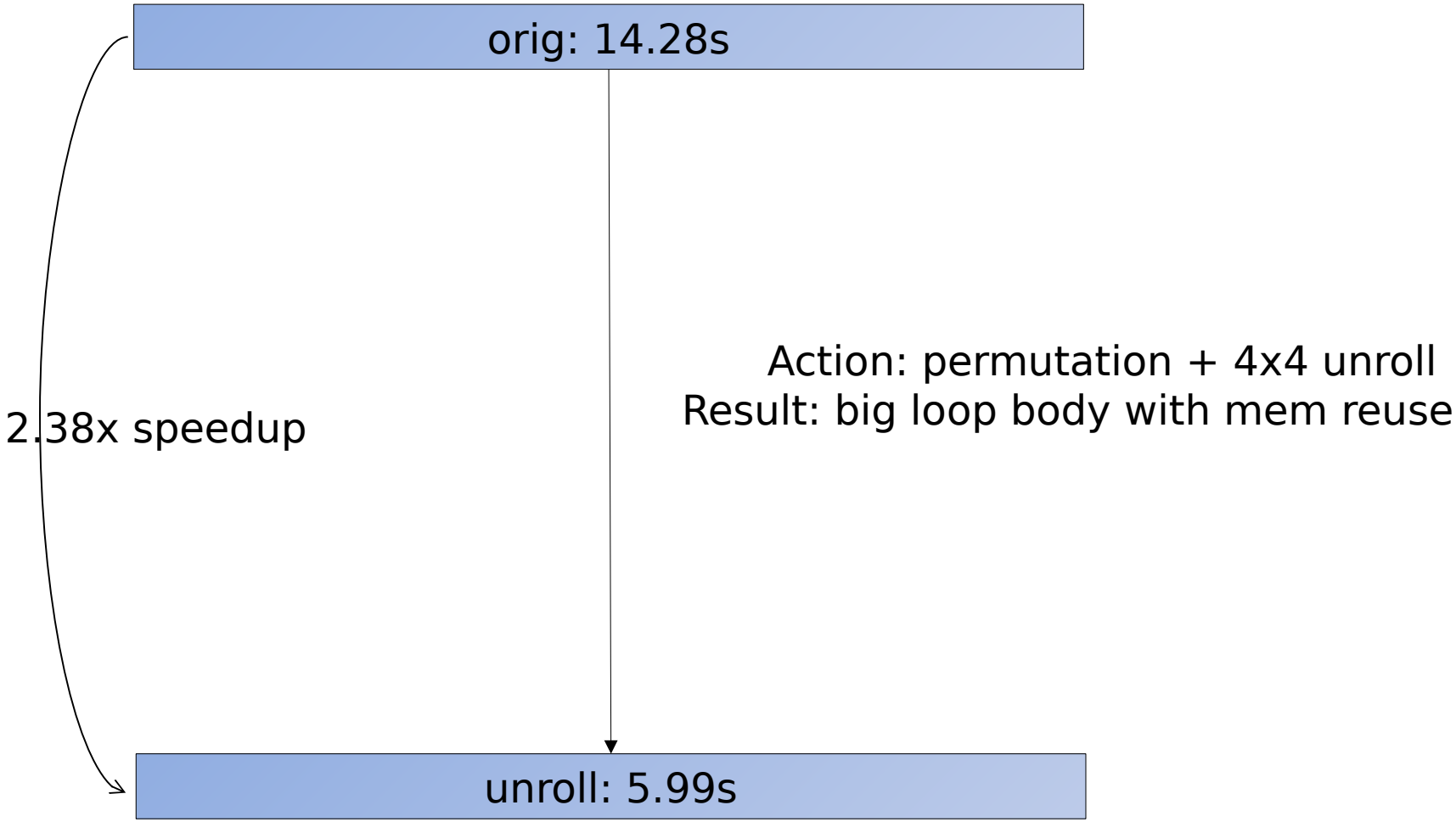
MAQAO Global Functions Loops

▶ Filters

Functions

Name	Module	Coverage (%)			Time (s)			Nb Threads			Deviation (coverage)			Deviation (time)		
		ov_orig	ov_perm	ov_unroll	ov_orig	ov_perm	ov_unroll	ov_orig	ov_perm	ov_unroll	ov_orig	ov_perm	ov_unroll	ov_orig	ov_perm	ov_
linearSolver	binary	92.68	92.04	82.79	13.23	11.97	4.96	1	1	1	0.00	0.00	0.00	0.00	0.00	0
advect.1.extracted	binary	3.68	4	8.77	0.52	0.52	0.53	1	1	1	0.00	0.00	0.00	0.00	0.00	0
c_velocitySolver	binary	1.75	1.88	4.43	0.25	0.25	0.26	1	1	1	0.00	0.00	0.00	0.00	0.00	0
project	binary	1.65	1.73	3.34	0.24	0.23	0.2	1	1	1	0.00	0.00	0.00	0.00	0.00	0
c_densitySolver	binary	0.18	0.08	0.5	0.02	0.01	0.03	1	1	1	0.00	0.00	0.00	0.00	0.00	0
setBoundry	binary	0.04	0.23	NA	0	0.03	NA	1	1	NA	0.00	0.00	NA	0.00	0.00	N
__aarch64_swp4_rel	libc-2.28.so	0.04	NA	0.08	0	NA	0	1	NA	1	0.00	NA	0.00	0.00	NA	0
__memset	libastring_aarch64.so	NA	NA	0.08	NA	NA	0	NA	NA	1	NA	NA	0.00	NA	NA	0
__aarch64_cas4_acq	libc-2.28.so	NA	0.04	NA	NA	0	NA	NA	1	NA	NA	0.00	NA	NA	0.00	N

Summary of optimizations and gains



More sample codes

More codes to study with MAQAO in

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