`matrix` Performance Report

20150608-190308

Abstract

This performance report is intended to support performance analysis and optimization activities. It includes details on program behavior, system configuration and capabilities, including support data from well-known performance analysis tools.

This report was generated using `hotspot` version `0.1`. Homepage `http://www.github.com/moreandres/hotspot`. Full execution log can be found at `~/.hotspot/matrix/20150608-190308/hotspot.log`.

Contents

1 Program

This section provides details about the program being analyzed.

1. Program: `matrix`.
   Program is the name of the program.
2. Timestamp: `20150608-190308`.
   Timestamp is a unique identifier used to store information on disk.
3. Parameters Range: `[2048, 2304, 2560, 2816, 3072, 3328, 3584, 3840]`.
   Parameters range is the problem size set used to scale the program.

2 System Capabilities

This section provides details about the system being used for the analysis.

2.1 System Configuration

This subsection provides details about the system configuration.

The hardware in the system is summarized using a hardware lister utility. It reports exact memory configuration, firmware version, mainboard configuration, CPU version and speed, cache configuration, bus speed and others.

The hardware configuration can be used to contrast the system capabilities to well-known benchmarks results on similar systems.

```
memory      7984MiB System memory
processor   Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz
bridge      440FX - 82441FX PMC [Natoma]
bridge      82371SB PIIX3 ISA [Natoma/Triton II]
storage     82371AB/EB/MB PIIX4 IDE
network     82540EM Gigabit Ethernet Controller
bridge      82371AB/EB/MB PIIX4 ACPI
storage     82801HM/HEM (ICH8M/ICH8M-E) SATA Controller [AHCI mode]
```

The software in the system is summarized using the GNU/Linux platform string.

```
Linux-3.16.0-30-generic-x86_64-with-Ubuntu-14.04-trusty
```

The software toolchain is built upon the following components.

1. Host: `ubuntu`

2. Distribution: `Ubuntu, 14.04, trusty`.
   This codename provides LSB (Linux Standard Base) and distribution-specific information.
3. Compiler: `gcc (Ubuntu 4.8.2-19ubuntu1) 4.8.2`.
   Version number of the compiler program.
4. C Library: `GNU C Library (Ubuntu EGLIBC 2.19-0ubuntu6.6) stable release version 2.19`.
   Version number of the C library.

The software configuration can be used to contrast the system capabilities to well-known benchmark results on similar systems.

2.2 System Performance Baseline

This subsection provides details about the system capabilities.

A set of performance results is included as a reference to contrast systems and to verify hardware capabilities using well-known synthetic benchmarks.

The HPC Challenge benchmark [1] consists of different tests:

1. HPL: the Linpack TPP benchmark which measures the floating point rate of execution for solving a linear system of equations.
2. DGEMM: measures the floating point rate of execution of double precision real matrix-matrix multiplication.
3. PTRANS (parallel matrix transpose): exercises the communications where pairs of processors communicate with each other simultaneously.
4. RandomAccess: measures the rate of integer random updates of memory (GUPS).
5. STREAM: a simple synthetic benchmark program that measures sustainable memory bandwidth (in GB/s).
6. FFT: measures the floating point rate of execution of double precision complex one-dimensional Discrete Fourier Transform (DFT).

Table 1: Benchmarks

Benchmark	**Value**	**Unit**
hpl	0.00385463 TFlops	tflops
dgemm	1.79842 GFlops	mflops
ptrans	1.8496 GBs	MB/s
random	0.0523812 GUPs	MB/s
stream	6.96329 MBs	MB/s
fft	2.03598 GFlops	MB/s

Most programs will have a dominant compute kernel that can be approximated by the ones above, the results helps to understand the available capacity.

3 Workload

This section provides details about the workload behavior.

3.1 Workload Footprint

The workload footprint impacts on memory hierarchy usage.

```
matrix: ELF 64-bit LSB  executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux
```

Binaries should be stripped to better fit inside cache.

```
struct _IO_FILE {
int                         _flags;               /*     0     4 */

/* XXX 4 bytes hole, try to pack */

char *                      _IO_read_ptr;         /*     8     8 */
char *                      _IO_read_end;         /*    16     8 */
char *                      _IO_read_base;        /*    24     8 */
char *                      _IO_write_base;       /*    32     8 */
char *                      _IO_write_ptr;        /*    40     8 */
char *                      _IO_write_end;        /*    48     8 */
char *                      _IO_buf_base;         /*    56     8 */
/* --- cacheline 1 boundary (64 bytes) --- */
char *                      _IO_buf_end;          /*    64     8 */
char *                      _IO_save_base;        /*    72     8 */
char *                      _IO_backup_base;      /*    80     8 */
```

```
char *                     _IO_save_end;         /*    88     8 */
struct _IO_marker *        _markers;             /*    96     8 */
struct _IO_FILE *          _chain;               /*   104     8 */
int                        _fileno;              /*   112     4 */
int                        _flags2;              /*   116     4 */
__off_t                    _old_offset;          /*   120     8 */
/* --- cacheline 2 boundary (128 bytes) --- */
short unsigned int         _cur_column;          /*   128     2 */
signed char                _vtable_offset;       /*   130     1 */
char                       _shortbuf[1];         /*   131     1 */

/* XXX 4 bytes hole, try to pack */

_IO_lock_t *               _lock;                /*   136     8 */
__off64_t                  _offset;              /*   144     8 */
void *                     __pad1;               /*   152     8 */
void *                     __pad2;               /*   160     8 */
void *                     __pad3;               /*   168     8 */
void *                     __pad4;               /*   176     8 */
size_t                     __pad5;               /*   184     8 */
/* --- cacheline 3 boundary (192 bytes) --- */
int                        _mode;                /*   192     4 */
char                       _unused2[20];         /*   196    20 */

/* size: 216, cachelines: 4, members: 29 */
/* sum members: 208, holes: 2, sum holes: 8 */
/* last cacheline: 24 bytes */
};
struct _IO_marker {
struct _IO_marker *        _next;                /*     0     8 */
struct _IO_FILE *          _sbuf;                /*     8     8 */
int                        _pos;                 /*    16     4 */

/* size: 24, cachelines: 1, members: 3 */
/* padding: 4 */
/* last cacheline: 24 bytes */
};
```

The in-memory layout of data structures can be used to identify issues. Reorganizing data to remove alignment holes will improve CPU cache utilization.

More information `https://www.kernel.org/doc/ols/2007/ols2007v2-pages-35-44.pdf`

3.2 Workload Stability

This subsection provides details about workload stability.

1. Execution time:
   (a) problem size range: `2048 - 4096`
   (b) geomean: `42.80017` seconds
   (c) average: `42.80048` seconds
   (d) stddev: `0.16483`
   (e) min: `42.41161` seconds
   (f) max: `43.15019` seconds
   (g) repetitions: `16` times

The histogram plots the elapsed times and shows how they fit in a normal distribution sample.

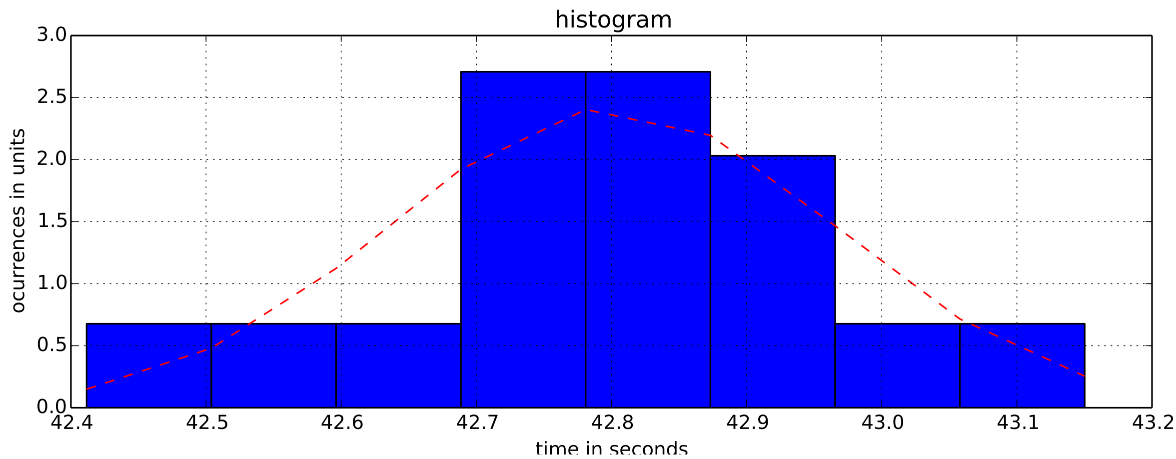

Figure 1: Results Distribution

The workload should run for at least one minute to fully utilize system resources. The execution time of the workload should be stable and the standard deviation less than 3 units.

3.3 Workload Optimization

This section shows how the program reacts to different optimization levels.

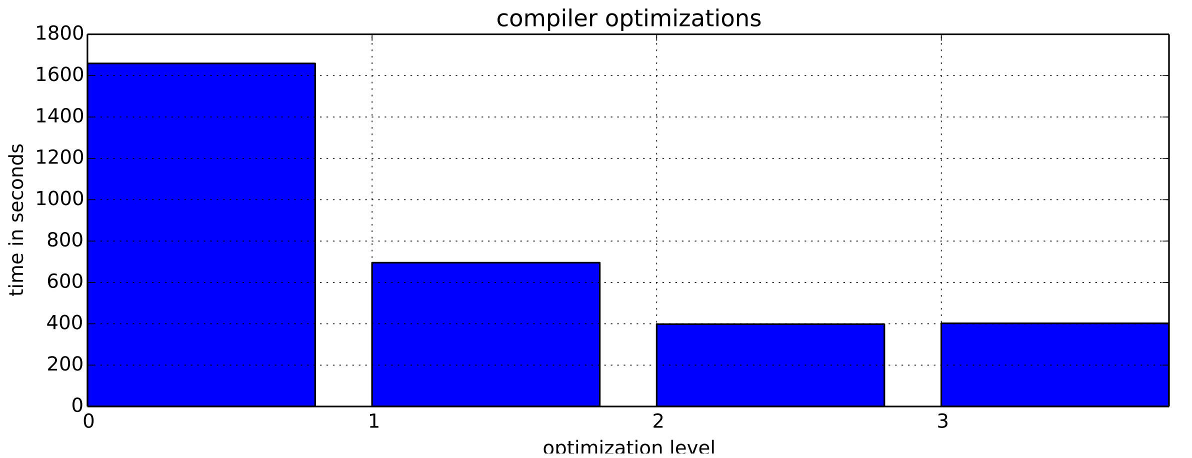

Figure 2: Optimization Levels

4 Scalability

This section provides details about the scaling behavior of the program.

4.1 Problem Size Scalability

A chart with the execution time when scaling the problem size.

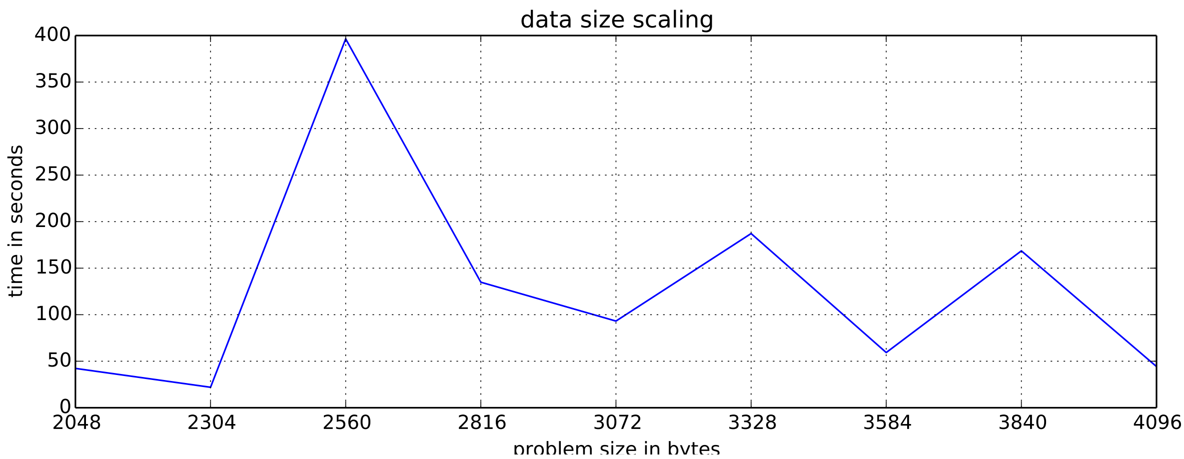

Figure 3: Problem size times

The chart will show how computing time increases when increasing problem size. There should be no valleys or bumps if processing properly balanced across computational units.

4.2 Computing Scalability

A chart with the execution time when scaling computation units.

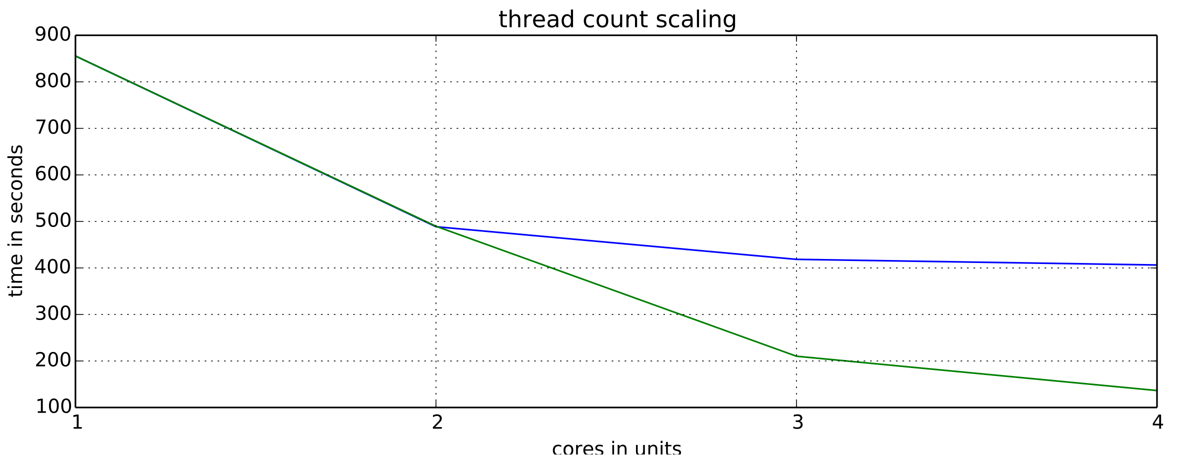

Figure 4: Thread count times

The chart will show how computing time decreases when increasing processing units. An ideal scaling line is provided for comparison.

The parallel and serial fractions of the program can be estimated using the information above.

1. Parallel Fraction: `0.85788`.
   Portion of the program doing parallel work.
2. Serial: `0.14212`.
   Portion of the program doing serial work.

Optimization limits can be estimated using scaling laws.

1. Amdalah Law for 1024 procs: `7.03614 times`.
   Optimizations are limited up to this point when scaling problem size. [2]
2. Gustafson Law for 1024 procs: `878.60771 times`.
   Optimizations are limited up to this point when not scaling problem size. [3]

5 Profile

This section provides details about the execution profile of the program and the system.

5.1 Program Profiling

This subsection provides details about the program execution profile.

5.1.1 Flat Profile

The flat profile shows how much time your program spent in each function, and how many times that function was called.

```
Flat profile:
Each sample counts as 0.01 seconds.
     cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 90.00    659.87   659.87                             main._omp_fn.0 (matrix.c:28 @ 400b48)
  9.96    732.90    73.03                             main._omp_fn.0 (matrix.c:28 @ 400b5b)
```

The table shows where to focus optimization efforts to maximize impact.

5.2 System Profiling

This subsection provide details about the system execution profile.

5.2.1 System Resources Usage

The following charts shows the state of system resources during the execution of the program.

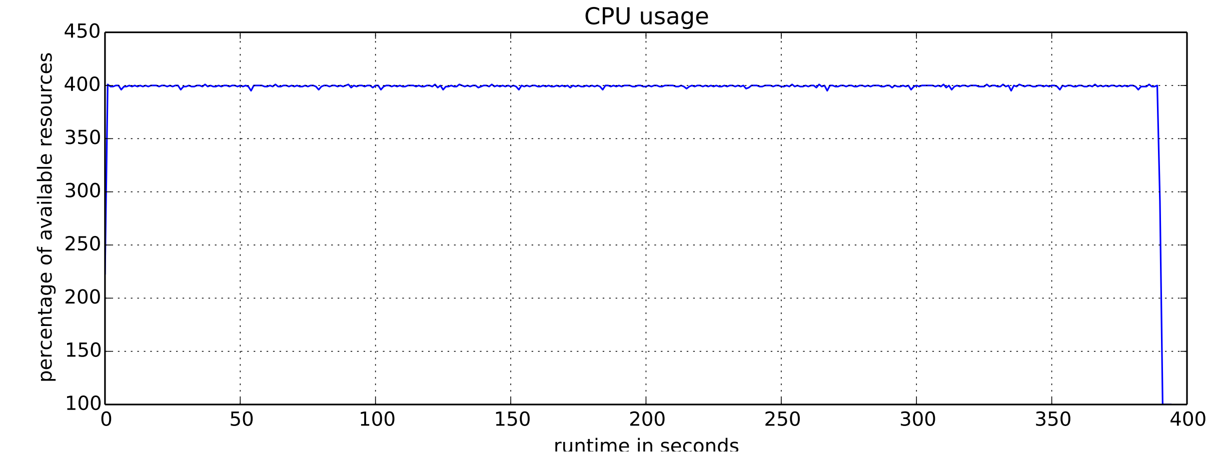

Figure 5: CPU Usage

Note that this chart is likely to show as upper limit a multiple of 100% in case a multicore system is being used.

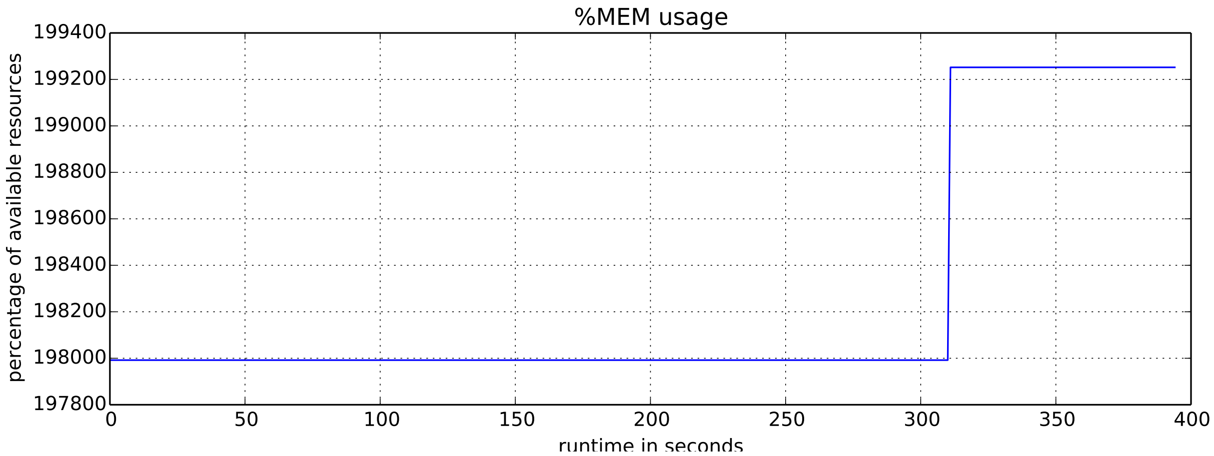

Figure 6: Memory Usage

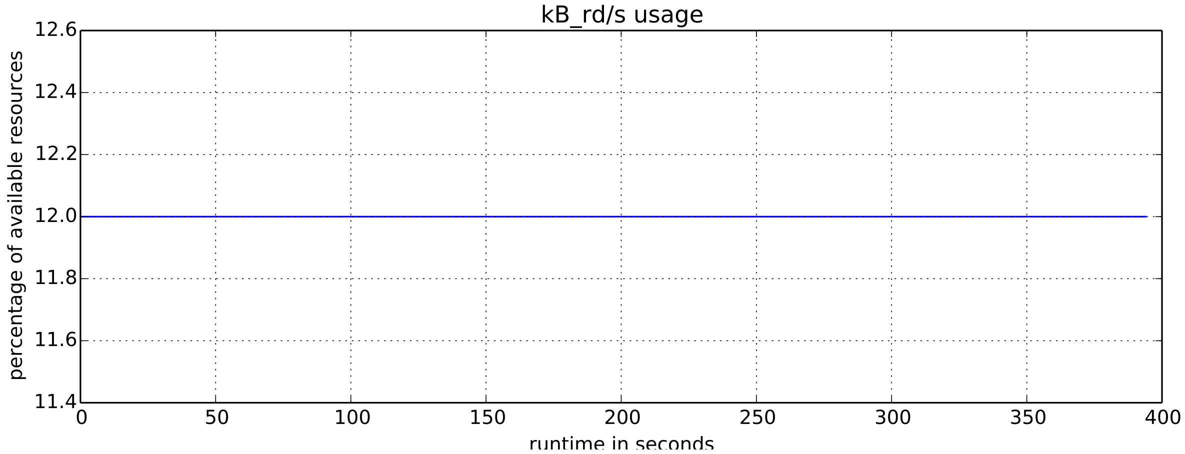

Figure 7: Reads from Disk

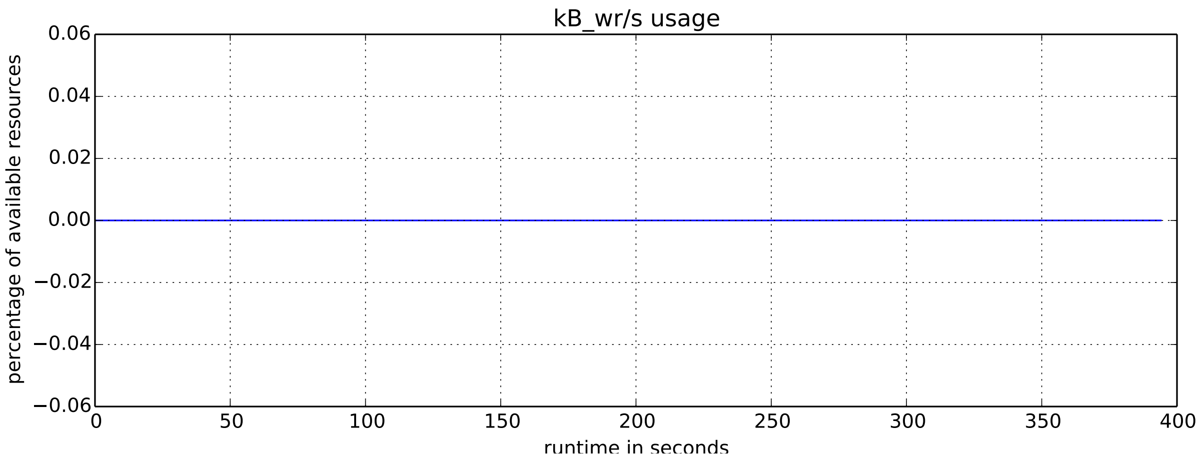

Figure 8: Writes to Disk

5.3 Hotspots

This subsection shows annotated code guiding the optimization efforts.

```
        : #pragma omp parallel for shared(a,b,c)
        :   for (i = 0; i < size; ++i) {
        :     for (j = 0; j < size; ++j) {
        :       for (k = 0; k < size; ++k) {
        :         c[i+j*size] += a[i+k*size] * b[k+j*size];
   0.20 :   4009e8:       movss  (rcx),xmm0
  77.69 :   4009ec:       add    r9,rcx
   9.81 :   4009ef:       mulss  (r8,rdx,4),xmm0
   6.46 :   4009f5:       add    $0x1,rdx
        :   }
        : #pragma omp parallel for shared(a,b,c)
        :   for (i = 0; i < size; ++i) {
        :     for (j = 0; j < size; ++j) {
        :       for (k = 0; k < size; ++k) {
        :         c[i+j*size] += a[i+k*size] * b[k+j*size];
   0.91 :   4009fb:       addss  xmm0,xmm1
   3.52 :   4009ff:       movss  xmm1,(rsi)
        :   }
        : #pragma omp parallel for shared(a,b,c)
        :   for (i = 0; i < size; ++i) {
        :     for (j = 0; j < size; ++j) {
        :       for (k = 0; k < size; ++k) {
--
        :       b[i+j*size] = (float) (i - j);
        :   int i, j, k;
        :   for (i = 0; i < size; ++i) {
        :     for (j = 0; j < size; ++j) {
```

```
         :       a[i+j*size] = (float) (i + j);
   64.82 :   4007c3:       cvtsi2ss edi,xmm0
         :   float *c = malloc(sizeof(float) * size * size);
         :   int i, j, k;
         :   for (i = 0; i < size; ++i) {
         :     for (j = 0; j < size; ++j) {
         :       a[i+j*size] = (float) (i + j);
    2.24 :   4007ce:       movss  xmm0,(r10,rcx,1)
         :       b[i+j*size] = (float) (i - j);
   29.06 :   4007d4:       cvtsi2ss edi,xmm0
    1.85 :   4007d8:       movss  xmm0,(r9,rcx,1)
    0.97 :   4007de:       add    r11,rcx
         :   float *c = malloc(sizeof(float) * size * size);
         :   int i, j, k;
         :   for (i = 0; i < size; ++i) {
         :     for (j = 0; j < size; ++j) {
         :   float *b = malloc(sizeof(float) * size * size);
```

6 Low Level

This section provide details about low level details such as vectorization and performance counters.

6.1 Vectorization Report

This subsection provide details about vectorization status of the program loops.

```
Analyzing loop at matrix.c:26
matrix.c:26: note: not vectorized: multiple nested loops.
matrix.c:26: note: bad loop form.
Analyzing loop at matrix.c:26
matrix.c:26: note: not vectorized: not suitable for strided load _41 = *_40;
matrix.c:26: note: bad data references.
Analyzing loop at matrix.c:27
matrix.c:27: note: step unknown.
matrix.c:27: note: reduction used in loop.
matrix.c:27: note: Unknown def-use cycle pattern.
matrix.c:27: note: Unsupported pattern.
matrix.c:27: note: not vectorized: unsupported use in stmt.
matrix.c:27: note: unexpected pattern.
matrix.c:24: note: vectorized 0 loops in function.
matrix.c:24: note: not consecutive access _10 = .omp_data_i_9(D)->size;
matrix.c:24: note: Failed to SLP the basic block.
matrix.c:24: note: not vectorized: failed to find SLP opportunities in basic block.
matrix.c:24: note: not vectorized: not enough data-refs in basic block.
matrix.c:24: note: not vectorized: not enough data-refs in basic block.
matrix.c:26: note: not vectorized: not enough data-refs in basic block.
matrix.c:24: note: not vectorized: not enough data-refs in basic block.
matrix.c:24: note: not consecutive access .omp_data_i_9(D)->j = .omp_data_i__j_lsm.9_106;
matrix.c:24: note: Failed to SLP the basic block.
matrix.c:24: note: not vectorized: failed to find SLP opportunities in basic block.
matrix.c:24: note: not vectorized: not enough data-refs in basic block.
matrix.c:24: note: Failed to SLP the basic block.
matrix.c:24: note: not vectorized: failed to find SLP opportunities in basic block.
matrix.c:24: note: not consecutive access pretmp_123 = *pretmp_122;
matrix.c:24: note: Failed to SLP the basic block.
matrix.c:24: note: not vectorized: failed to find SLP opportunities in basic block.
matrix.c:26: note: not vectorized: not enough data-refs in basic block.
matrix.c:24: note: not consecutive access .omp_data_i_9(D)->k = _10;
matrix.c:24: note: Failed to SLP the basic block.
matrix.c:24: note: not vectorized: failed to find SLP opportunities in basic block.
matrix.c:24: note: not vectorized: not enough data-refs in basic block.
matrix.c:24: note: not vectorized: not enough data-refs in basic block.
matrix.c:28: note: can't determine dependence between *_40 and *pretmp_122
matrix.c:28: note: can't determine dependence between *_46 and *pretmp_122
matrix.c:28: note: SLP: step doesn't divide the vector-size.
matrix.c:28: note: Unknown alignment for access: *(pretmp_113 + (sizetype) ((long unsigned int) pretmp_118 * 4))
matrix.c:28: note: not consecutive access _41 = *_40;
matrix.c:28: note: not consecutive access *pretmp_122 = _49;
matrix.c:28: note: Failed to SLP the basic block.
```

```
matrix.c:28: note: not vectorized: failed to find SLP opportunities in basic block.
matrix.c:24: note: not vectorized: not enough data-refs in basic block.
matrix.c:24: note: not vectorized: not enough data-refs in basic block.
Analyzing loop at matrix.c:16
matrix.c:16: note: not vectorized: not suitable for strided load *_27 = _29;
matrix.c:16: note: bad data references.
Analyzing loop at matrix.c:17
matrix.c:17: note: not vectorized: not suitable for strided load *_27 = _29;
matrix.c:17: note: bad data references.
matrix.c:4: note: vectorized 0 loops in function.
matrix.c:7: note: not vectorized: not enough data-refs in basic block.
matrix.c:8: note: not vectorized: not enough data-refs in basic block.
matrix.c:4: note: not vectorized: not enough data-refs in basic block.
matrix.c:18: note: not consecutive access *_27 = _29;
matrix.c:18: note: not consecutive access *_32 = _34;
matrix.c:18: note: not consecutive access *_36 = 0.0;
matrix.c:18: note: Failed to SLP the basic block.
matrix.c:18: note: not vectorized: failed to find SLP opportunities in basic block.
matrix.c:16: note: not vectorized: not enough data-refs in basic block.
matrix.c:4: note: not vectorized: not enough data-refs in basic block.
matrix.c:4: note: not vectorized: not enough data-refs in basic block.
matrix.c:24: note: misalign = 0 bytes of ref .omp_data_o.1.c
matrix.c:24: note: misalign = 8 bytes of ref .omp_data_o.1.b
matrix.c:24: note: misalign = 0 bytes of ref .omp_data_o.1.a
matrix.c:24: note: misalign = 8 bytes of ref .omp_data_o.1.size
matrix.c:24: note: misalign = 12 bytes of ref .omp_data_o.1.j
matrix.c:24: note: misalign = 0 bytes of ref .omp_data_o.1.k
matrix.c:24: note: Build SLP failed: unrolling required in basic block SLP
matrix.c:24: note: Build SLP failed: unrolling required in basic block SLP
matrix.c:24: note: Failed to SLP the basic block.
matrix.c:24: note: not vectorized: failed to find SLP opportunities in basic block.
matrix.c:4: note: not vectorized: not enough data-refs in basic block.
matrix.c:10: note: not vectorized: not enough data-refs in basic block.
matrix.c:4: note: not vectorized: not enough data-refs in basic block.
```

The details above shows the list of loops in the program and if they are being vectorized or not. These reports can pinpoint areas where the compiler cannot apply vectorization and related optimizations. It may be possible to modify your code or communicate additional information to the compiler to guide the vectorization and/or optimizations.

6.2 Counters Report

This subsection provides details about software and hardware counters.

```
 Performance counter stats for './matrix' (3 runs):

    1568409.582702      task-clock (msec)        #    3.954 CPUs utilized            ( +-  0.16 )
             2,863      context-switches         #    0.002 K/sec                    ( +-  0.80 )
                 3      cpu-migrations           #    0.000 K/sec                    ( +- 10.00 )
             1,703      page-faults              #    0.001 K/sec                    ( +-  0.04 )
   <not supported>      cycles
   <not supported>      stalled-cycles-frontend
   <not supported>      stalled-cycles-backend
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

     396.643251906 seconds time elapsed                                          ( +-  0.31 )
```

The details above shows counters that provide low-overhead access to detailed performance information using internal registers of the CPU.

References

[1] Piotr Luszczek and Jack J. Dongarra and David Koester and Rolf Rabenseifner and Bob Lucas and Jeremy Kepner and John Mccalpin and David Bailey and Daisuke Takahashi, *Introduction to the HPC Challenge Benchmark Suite*. Technical Report, 2005.

[2] Amdahl, Gene M., *Validity of the single processor approach to achieving large scale computing capabilities*. Communications of the ACM, Proceedings of the April 18-20, 1967, spring joint computer conference Pages 483-485, 1967.

[3] John L. Gustafson, *Reevaluating Amdahl's Law*. Communications of the ACM, Volume 31 Pages 532-533, 1988.

[4] OpenMP Architecture Review Board, *OpenMP Application Program Interface. http://www.openmp.org*, 3.0, May 2008.