

heat2d Performance Report

20140913-174730

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/heat2d/20140913-174730/hotspot.log`.

Contents

<h1>Contents</h1>			4.1	Problem Size Scalability	4
1	Program	1	4.2	Computing Scalability	5
2	System Capabilities	1	5	Profile	5
2.1	System Configuration	1	5.1	Program Profiling	5
2.2	System Performance Baseline	2	5.1.1	Flat Profile	6
3	Workload	2	5.2	System Profiling	6
3.1	Workload Footprint	2	5.2.1	System Resources Usage	6
3.2	Workload Stability	3	5.3	Hotspots	7
3.3	Workload Optimization	4	6	Low Level	8
4	Scalability	4	6.1	Vectorization Report	8
			6.2	Counters Report	10

1 Program

This section provides details about the program being analyzed.

1. Program: `heat2d`.
Program is the name of the program.
2. Timestamp: `20140913-174730`.
Timestamp is a unique identifier used to store information on disk.
3. Parameters Range: `[4096, 4608, 5120, 5632, 6144, 6656, 7168, 7680]`.
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      7985MiB System memory
processor    Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz
bridge      440FX - 82441FX PMC [Natoma]
bridge      82371SB PIIIX3 ISA [Natoma/Triton II]
storage     82371AB/EB/MB PIIIX4 IDE
network     82540EM Gigabit Ethernet Controller
bridge      82371AB/EB/MB PIIIX4 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-3.13.0-32-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 2.19-0ubuntu6.3) 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	0.00343933 TFlops	tflops
dgemm	0.939712 GFlops	mflops
ptrans	0.373827 GBs	MB/s
random	0.0275466 GUPs	MB/s
stream	5.34933 MBs	MB/s
fft	1.06575 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.

heat2d: 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 */
};

```

Showing layout of data structures. Reorganizing such data to remove alignment holes. Improve CPU cache utilization.

More information [sevendwarveshttps://www.kernel.org/doc/ols/2007/ols2007v2-pages-35-44.pdf](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: 4096 - 8192
 - (b) geomean: 5.16793 seconds
 - (c) average: 5.17504 seconds
 - (d) stddev: 0.27067
 - (e) min: 4.80759 seconds
 - (f) max: 5.50437 seconds
 - (g) repetitions: 8 times

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

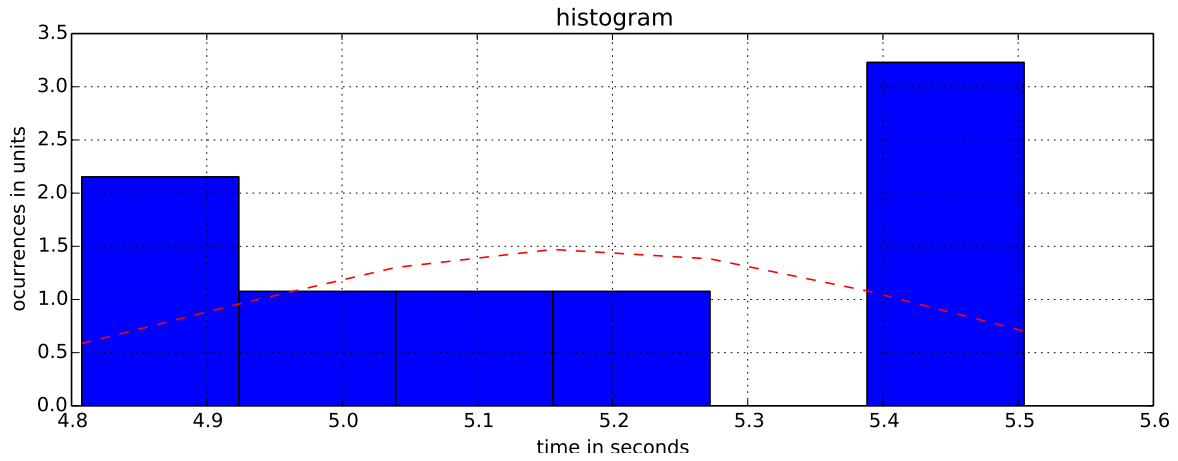


Figure 1: Results Distribution

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.

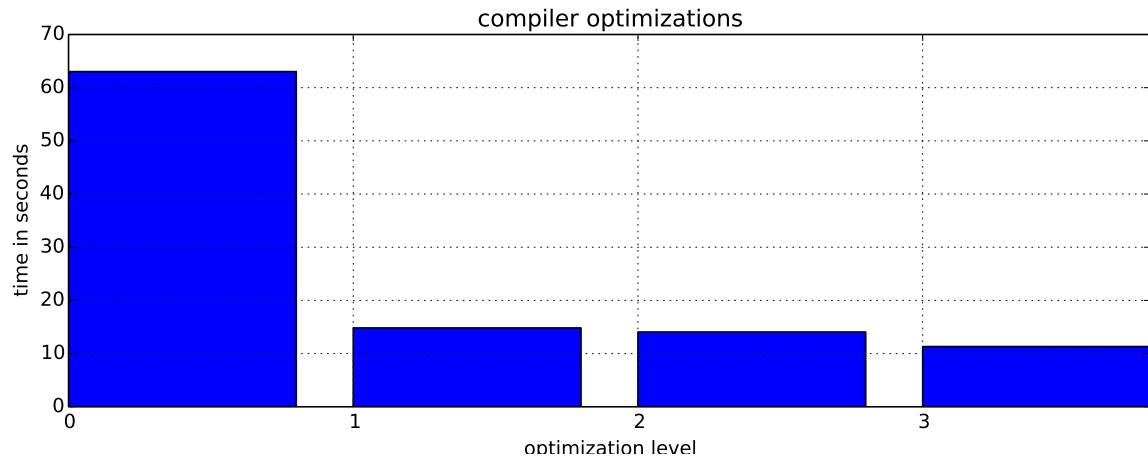


Figure 2: Optimization Levels

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.

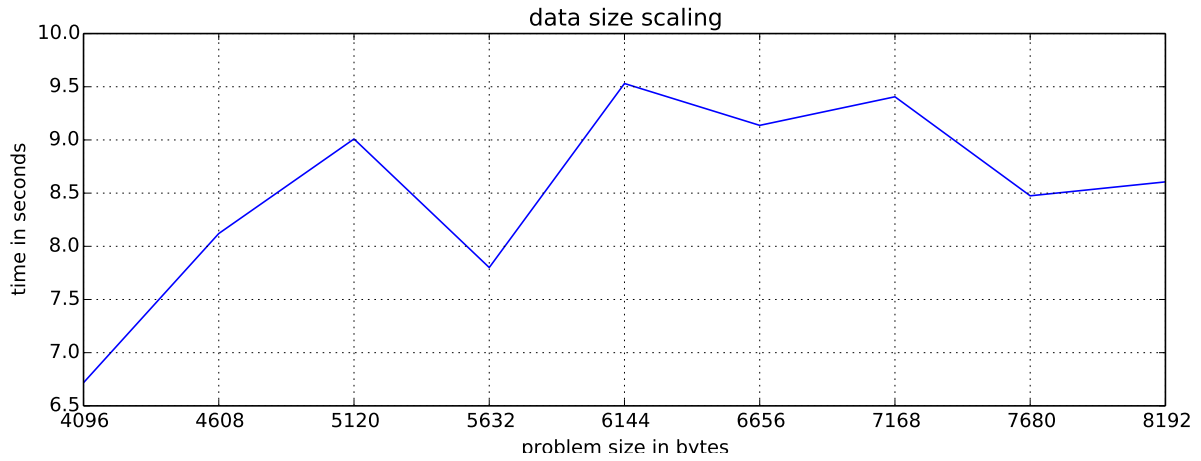


Figure 3: Problem size times

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.

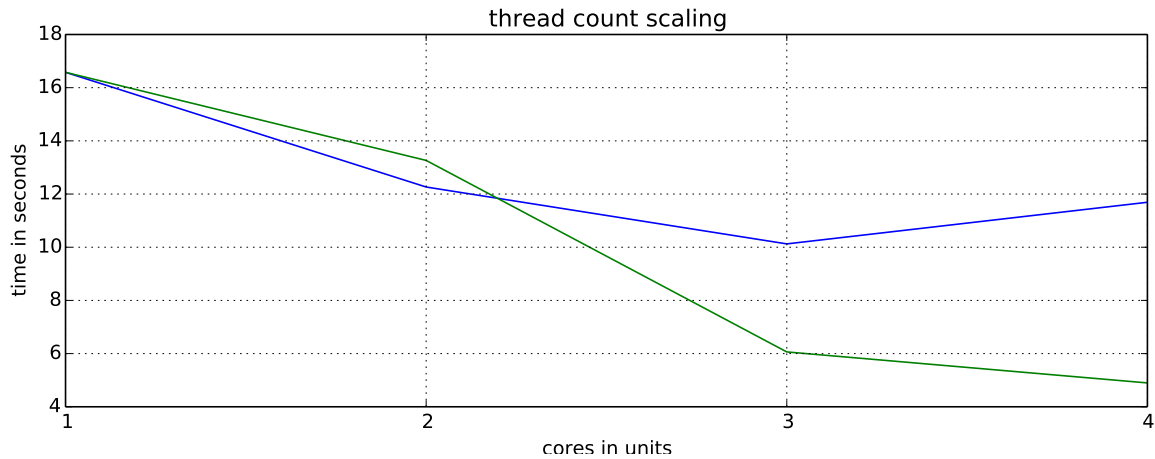


Figure 4: Thread count times

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: 0.52039.
Portion of the program doing parallel work.
2. Serial: 0.47961.
Portion of the program doing serial work.

Optimization limits can be estimated using scaling laws.

1. Amdalah Law for 1024 procs: 2.08504 times.
Optimizations are limited up to this point when scaling problem size. [?]
2. Gustafson Law for 1024 procs: 533.36271 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
26.27	3.49	3.49				compute_one_iteration._omp_fn.1 (heat2d.c:78 @ 400d97)
13.29	5.26	1.77				compute_one_iteration._omp_fn.1 (heat2d.c:83 @ 400d93)
9.29	6.50	1.24				compute_one_iteration._omp_fn.1 (heat2d.c:82 @ 400d8f)
9.14	7.71	1.21				compute_one_iteration._omp_fn.1 (heat2d.c:83 @ 400d70)
9.14	8.93	1.21				compute_one_iteration._omp_fn.1 (heat2d.c:75 @ 400bd0)
7.55	9.93	1.00				compute_one_iteration._omp_fn.1 (heat2d.c:79 @ 400d50)
7.40	10.91	0.98				compute_one_iteration._omp_fn.1 (heat2d.c:81 @ 400d89)
4.68	11.54	0.62				compute_one_iteration._omp_fn.1 (heat2d.c:80 @ 400d69)
4.53	12.14	0.60				compute_one_iteration._omp_fn.1 (heat2d.c:80 @ 400d83)
3.77	12.64	0.50				compute_one_iteration._omp_fn.1 (heat2d.c:83 @ 400d63)
2.57	12.98	0.34				compute_one_iteration._omp_fn.1 (heat2d.c:83 @ 400d7b)
1.81	13.22	0.24				compute_one_iteration._omp_fn.1 (heat2d.c:79 @ 400d5c)

Call graph

granularity: each sample hit covers 2 byte(s) for 0.07 of 13.35 seconds

index	time	self	children	called	name
					<spontaneous>
[15]	0.1	0.01	0.00		compute_one_iteration (heat2d.c:54 @ 400ee0) [15]
[20]	0.0	0.00	0.00	74410	frame_dummy [20]

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.

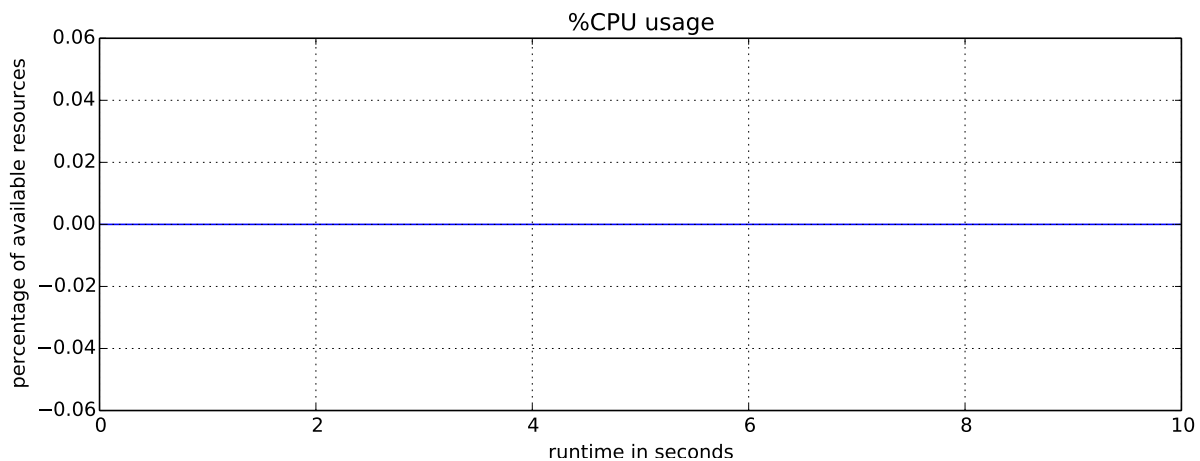


Figure 5: CPU Usage

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

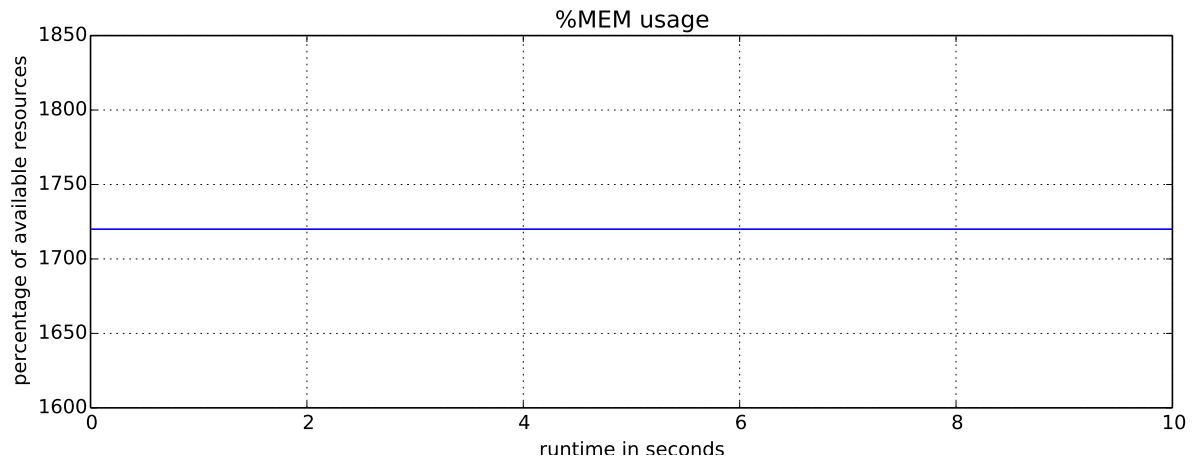


Figure 6: Memory Usage

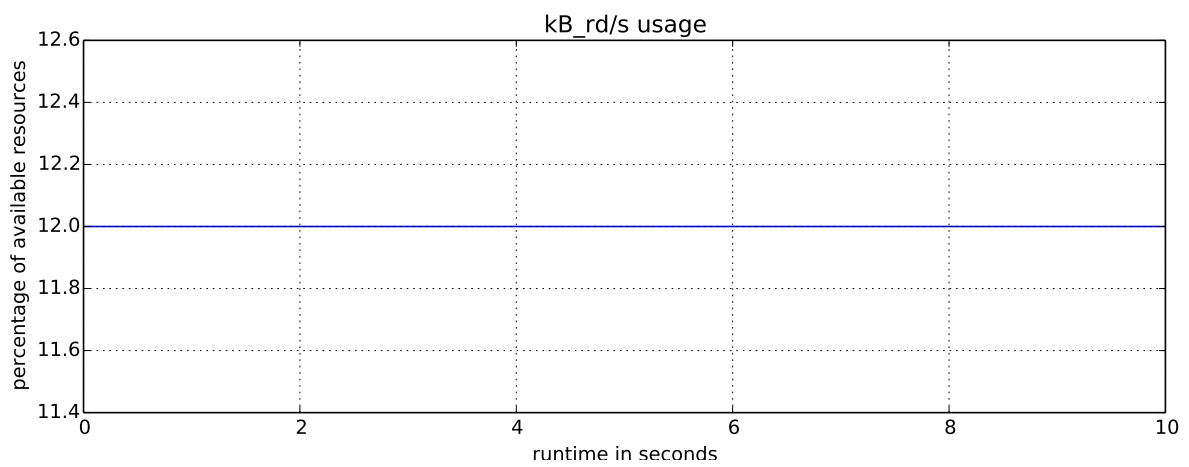


Figure 7: Reads from Disk

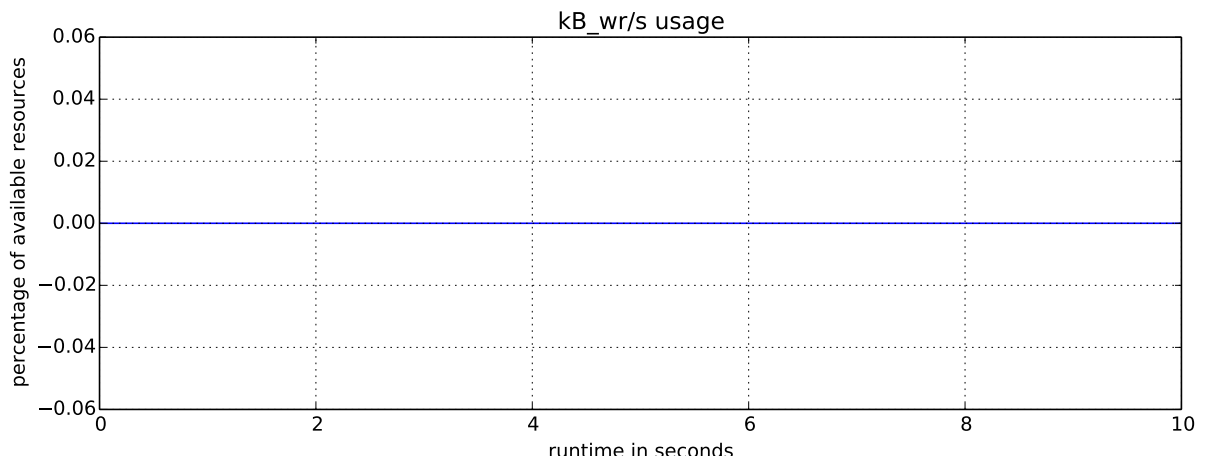


Figure 8: Writes to Disk

5.3 Hotspots

This subsection shows annotated code guiding the optimization efforts.

```
[kernel.kallsyms] with build id fa854edfe97c596cddeac7049b080cbbba9de2775 not found, continuing without symbols
:      {
:      if (i < 256 || i > 768)
Percent | Source code & Disassembly of libc-2.19.so
-----
: Disassembly of section .text:
36.36 : 83877:      mov    (rax),rax
```

```

18.18 : 8387a:      test    rax,rax
-----
      : Disassembly of section .text:
33.33 : 74e3:      retq
Percent | Source code & Disassembly of libc-2.19.so
-----
      : Disassembly of section .text:
-----
      : Disassembly of section .text:

```

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 heat2d.c:46

heat2d.c:46: note: not vectorized: loop contains function calls or data references that cannot be analyzed

heat2d.c:46: note: bad data references.

heat2d.c:43: note: vectorized 0 loops in function.

heat2d.c:43: note: not vectorized: not enough data-refs in basic block.

heat2d.c:43: note: not vectorized: not enough data-refs in basic block.

heat2d.c:43: note: not vectorized: not enough data-refs in basic block.

heat2d.c:46: note: not vectorized: not enough data-refs in basic block.

heat2d.c:43: note: not vectorized: not enough data-refs in basic block.

heat2d.c:43: note: not vectorized: not enough data-refs in basic block.

heat2d.c:43: note: not vectorized: not enough data-refs in basic block.

heat2d.c:75: note: versioning for alias required: can't determine dependence between solution[pretmp_108][pretmp_108]

heat2d.c:75: note: versioning not yet supported for outer-loops.

heat2d.c:75: note: bad data dependence.

Analyzing loop at heat2d.c:77

heat2d.c:77: note: versioning for alias required: can't determine dependence between solution[pretmp_108][pretmp_108]

heat2d.c:77: note: versioning for alias required: can't determine dependence between solution[pretmp_108][pretmp_108]

heat2d.c:77: note: versioning for alias required: can't determine dependence between solution[pretmp_108][i_3][j_29]

heat2d.c:77: note: versioning for alias required: can't determine dependence between solution[pretmp_108][i_3][j_29]

heat2d.c:77: note: versioning for alias required: can't determine dependence between solution[pretmp_108][i_3][j_29]

heat2d.c:77: note: misalign = 8 bytes of ref solution[pretmp_108][pretmp_112][j_39]

heat2d.c:77: note: misalign = 8 bytes of ref solution[pretmp_108][pretmp_113][j_39]

heat2d.c:77: note: misalign = 0 bytes of ref solution[pretmp_108][i_3][j_26]

heat2d.c:77: note: misalign = 0 bytes of ref solution[pretmp_108][i_3][j_29]

heat2d.c:77: note: misalign = 8 bytes of ref solution[pretmp_108][i_3][j_39]

heat2d.c:77: note: misalign = 8 bytes of ref solution[pretmp_106][i_3][j_39]

heat2d.c:77: note: num. args = 4 (not unary/binary/ternary op).

heat2d.c:77: note: not ssa-name.

heat2d.c:77: note: use not simple.

heat2d.c:77: note: num. args = 4 (not unary/binary/ternary op).

heat2d.c:77: note: not ssa-name.

heat2d.c:77: note: use not simple.

heat2d.c:77: note: num. args = 4 (not unary/binary/ternary op).

heat2d.c:77: note: not ssa-name.

heat2d.c:77: note: use not simple.

heat2d.c:77: note: num. args = 4 (not unary/binary/ternary op).

heat2d.c:77: note: not ssa-name.

heat2d.c:77: note: use not simple.

heat2d.c:77: note: num. args = 4 (not unary/binary/ternary op).

heat2d.c:77: note: not ssa-name.

heat2d.c:77: note: use not simple.

Vectorizing loop at heat2d.c:77

heat2d.c:75: note: vectorized 1 loops in function.

heat2d.c:75: note: not vectorized: not enough data-refs in basic block.

heat2d.c:75: note: not vectorized: not enough data-refs in basic block.

heat2d.c:75: note: not consecutive access pretmp_106 = next_gen;

heat2d.c:75: note: not consecutive access pretmp_108 = cur_gen;

heat2d.c:75: note: not consecutive access pretmp_110 = diff_constant;

heat2d.c:75: note: Failed to SLP the basic block.

heat2d.c:75: note: not vectorized: failed to find SLP opportunities in basic block.

heat2d.c:75: note: not vectorized: not enough data-refs in basic block.


```

heat2d.c:75: note: not vectorized: not enough data-refs in basic block.
heat2d.c:75: note: not vectorized: not enough data-refs in basic block.
heat2d.c:75: note: not vectorized: not enough data-refs in basic block.
heat2d.c:75: note: not vectorized: not enough data-refs in basic block.
heat2d.c:79: note: can't determine dependence between solution[pretmp_108][pretmp_112][j_140] and solution[pretmp_108][pretmp_113][j_140]
heat2d.c:79: note: can't determine dependence between solution[pretmp_108][pretmp_113][j_140] and solution[pretmp_106][i_3][j_146]
heat2d.c:79: note: can't determine dependence between solution[pretmp_108][i_3][j_146] and solution[pretmp_106][i_3][j_149]
heat2d.c:79: note: can't determine dependence between solution[pretmp_108][i_3][j_140] and solution[pretmp_106][i_3][j_146]
heat2d.c:79: note: SLP: step doesn't divide the vector-size.
heat2d.c:79: note: Unknown alignment for access: solution
heat2d.c:79: note: SLP: step doesn't divide the vector-size.
heat2d.c:79: note: Unknown alignment for access: solution
heat2d.c:79: note: SLP: step doesn't divide the vector-size.
heat2d.c:79: note: Unknown alignment for access: solution
heat2d.c:79: note: SLP: step doesn't divide the vector-size.
heat2d.c:79: note: Unknown alignment for access: solution
heat2d.c:79: note: SLP: step doesn't divide the vector-size.
heat2d.c:79: note: Unknown alignment for access: solution
heat2d.c:79: note: Failed to SLP the basic block.
heat2d.c:79: note: not vectorized: failed to find SLP opportunities in basic block.
heat2d.c:75: note: not vectorized: not enough data-refs in basic block.
heat2d.c:75: note: not vectorized: not enough data-refs in basic block.
heat2d.c:79: note: not vectorized: no vectype for stmt: vect_var_.72_169 = MEM[(double[2][302][302] *)vect_psolution]
scalar_type: vector(2) double
heat2d.c:79: note: Failed to SLP the basic block.
heat2d.c:79: note: not vectorized: failed to find SLP opportunities in basic block.
heat2d.c:75: note: not vectorized: not enough data-refs in basic block.
heat2d.c:75: note: not vectorized: not enough data-refs in basic block.
heat2d.c:43: note: not vectorized: not enough data-refs in basic block.
Analyzing loop at heat2d.c:64
heat2d.c:64: note: misalign = 0 bytes of ref solution[pretmp_34][i_41][300]
heat2d.c:64: note: misalign = 8 bytes of ref solution[pretmp_34][i_41][301]
heat2d.c:64: note: not consecutive access _19 = solution[pretmp_34][i_41][300];
heat2d.c:64: note: not vectorized: complicated access pattern.
heat2d.c:64: note: bad data access.
Analyzing loop at heat2d.c:57
heat2d.c:57: note: not vectorized: control flow in loop.
heat2d.c:57: note: bad loop form.
heat2d.c:53: note: vectorized 0 loops in function.
heat2d.c:53: note: not consecutive access pretmp_34 = cur_gen;
heat2d.c:53: note: Failed to SLP the basic block.
heat2d.c:53: note: not vectorized: failed to find SLP opportunities in basic block.
heat2d.c:59: note: not vectorized: not enough data-refs in basic block.
heat2d.c:60: note: misalign = 8 bytes of ref solution[pretmp_34][i_40][1]
heat2d.c:60: note: misalign = 0 bytes of ref solution[pretmp_34][i_40][0]
heat2d.c:60: note: not consecutive access _12 = solution[pretmp_34][i_40][1];
heat2d.c:60: note: not consecutive access solution[pretmp_34][i_40][0] = _12;
heat2d.c:60: note: Failed to SLP the basic block.
heat2d.c:60: note: not vectorized: failed to find SLP opportunities in basic block.
heat2d.c:62: note: misalign = 0 bytes of ref solution[pretmp_34][i_40][0]
heat2d.c:62: note: not consecutive access solution[pretmp_34][i_40][0] = 2.0e+1;
heat2d.c:62: note: Failed to SLP the basic block.
heat2d.c:62: note: not vectorized: failed to find SLP opportunities in basic block.
heat2d.c:57: note: not vectorized: not enough data-refs in basic block.
heat2d.c:53: note: not vectorized: not enough data-refs in basic block.
heat2d.c:53: note: not vectorized: not enough data-refs in basic block.
heat2d.c:66: note: misalign = 0 bytes of ref solution[pretmp_34][i_41][300]
heat2d.c:66: note: misalign = 8 bytes of ref solution[pretmp_34][i_41][301]
heat2d.c:66: note: not consecutive access _19 = solution[pretmp_34][i_41][300];
heat2d.c:66: note: not consecutive access solution[pretmp_34][i_41][301] = _19;
heat2d.c:66: note: Failed to SLP the basic block.
heat2d.c:66: note: not vectorized: failed to find SLP opportunities in basic block.
heat2d.c:53: note: not vectorized: not enough data-refs in basic block.
heat2d.c:70: note: not vectorized: not enough data-refs in basic block.
Analyzing loop at heat2d.c:28
heat2d.c:28: note: not vectorized: number of iterations cannot be computed.
heat2d.c:28: note: bad loop form.

```

```

heat2d.c:18: note: vectorized 0 loops in function.
heat2d.c:20: note: misalign = 0 bytes of ref final
heat2d.c:20: note: not consecutive access final = 1.024e+3;
heat2d.c:20: note: Failed to SLP the basic block.
heat2d.c:20: note: not vectorized: failed to find SLP opportunities in basic block.
heat2d.c:22: note: not vectorized: not enough data-refs in basic block.
heat2d.c:26: note: not vectorized: not enough data-refs in basic block.
heat2d.c:36: note: not vectorized: not enough data-refs in basic block.
heat2d.c:18: note: not vectorized: not enough data-refs in basic block.
heat2d.c:30: note: not vectorized: not enough data-refs in basic block.
heat2d.c:18: 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 './heat2d' (3 runs):

43611.260053	task-clock (msec)	#	3.700 CPUs utilized	(+- 3.62)
2,112	context-switches	#	0.048 K/sec	(+- 12.35)
12	cpu-migrations	#	0.000 K/sec	(+- 18.74)
538	page-faults	#	0.012 K/sec	(+- 0.06)
<not supported>	cycles			
<not supported>	stalled-cycles-frontend			
<not supported>	stalled-cycles-backend			
<not supported>	instructions			
<not supported>	branches			
<not supported>	branch-misses			
11.785455878	seconds time elapsed			(+- 4.91)

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.