Using MACPO for performance analysis

MACPO is an acronym for Memory Access Centric Performance Optimization. It is a tool that has been built to assist performance tuning of single- and multi-threaded C, C++ or Fortran applications. More specifically, MACPO is designed to provide insight into an application's memory usage patterns.

1. A small use-case

To demonstrate the functioning of MACPO, let's consider an example program. This program uses Pthreads to calculate the value of Pi using Monte-Carlo method [http://www.mathcs.emory.edu/~cheung/Courses/170/Syllabus/07/compute-pi.html].

The following code shows the function that is executed by each thread. The full source code, with a Makefile, is available at http://goo.gl/uEVrh.

```
void* thread_func (void* arg) {
	int t;
	float x, y, z;
	thread_info_t* thread_info = (thread_info_t*) arg;
	for (int repeat=0; repeat<REPEAT_COUNT; repeat++) {
		srand(time(NULL) + thread_info->tid);
		for (int i=0; i<ITERATIONS; i++) {
			t = i + thread_info->tid;
			x = random_numbers[t % RANDOM_BUFFER_SIZE];
			y = random_numbers[(1 + t) % RANDOM_BUFFER_SIZE];

			z = x*x + y*y;
			if (z <= 1) counts[thread_info->tid]++;
		}
	}

	pthread_exit(0);
}
```

To compile the application using MACPO, indicating that we are interested in understanding the performance metrics associated with the `thread_func` function, run the following commands:

```
$ macpo.sh --macpo:function=thread_func monte-carlo.c -o monte-carlo -lpthread -lrt
```

Runtime logs (macpo.out) are produced when the application is run:

```
$ ./monte-carlo
```

To print performance metrics, use the macpo-analyze program.

```
$ macpo-analyze macpo.out
```

This will produce an output that is similar to the one shown below:

```
Var "counts", seen 1668 times, estimated to cost 147.12 cycles on every access
Stride of 0 cache lines was observed 1585 times (100.00%).

Level 1 data cache conflicts = 78.22% [################################       ]
Level 2 data cache conflicts = 63.37% [#########################              ]
NUMA data conflicts = 43.56%          [#################                      ]

Level 1 data cache reuse factor = 97.0% [######################################## ]
Level 2 data cache reuse factor = 3.0% [##                                       ]
Level 3 data cache reuse factor = 0.0% [                                         ]
```

The output shows that accesses to the variable `counts` suffer from cache conflicts (also known as cache thrashing [http://en.wikipedia.org/wiki/Thrashing_(computer_science)]). Using this knowledge, we can pad the `counts` array (i.e. add dummy bytes to the array) so that each thread accesses a different cache line. The optimized program is included in the tarball archive [http://goo.gl/uEVrh]. This optimization reduces the running time of the `thread_func` routine from 9.14s to 3.17s.

2. When to use MACPO

MACPO may be useful to you in any of the following situations:

- Perfexpert reports that L2 or L3 data access LCPI is high.
- Your program uses a lot of memory or if it is memory-bound
- CPU profiling does not show any interesting results.

If all (or most) of your application's memory accesses are irregular, you may be able to infer the optimizations applicable to your program. However, for such programs, MACPO metrics may

not be able to assist you directly.

3. MACPO metrics

Performance information shown by MACPO can be grouped into two parts. The first part shows information that is applicable to the entire function being profiled. The second part shows information that is specific to the important variables in the function.

3.1 Function-wide performance metrics

Currently, function-wide information only shows the number of streams that were seen while compiling the function. A stream is variable or data structure that may be accessed repeatedly in a uniform manner.

3.2 Variable-specific performance metrics

For each variable that is encountered a significant number of times, MACPO shows:

- Estimated average cycles per access
- Dominant stride values and their percentages
- Cache conflicts (thrashing) for level 1 and level 2 data caches
- NUMA conflicts
- Reuse factors for level 1, 2 and 3 data caches

The following sections explain each of the above metrics.

3.2.1 Estimated average cycles per accesses:

MACPO collects metrics that allow it to calculate the approximate reuse distance of variables. The reuse distance, in turn, helps to estimate the specific level of the cache that the request would have originated from. This makes it possible to estimate the number of cycles spent in each memory access. This cost in terms of memory access is grouped by the variable name and averaged to show in this metric.

3.2.2 Dominant stride values and percentages:

A stride is the constant difference in bytes between the last memory access and the most recent memory access to a variable. MACPO computes the access strides in units of cache line size.

3.2.3 Cache conflicts (thrashing) for level 1 and level 2 data caches:

Cache conflicts arise when multiple processors, each with at least one private cache, repeatedly claim exclusive access to a portion of memory. This metric shows the percentage of requests to each level of the cache that conflicted with another access to the same variable.

3.2.4 NUMA conflicts:

Most modern processors exhibit non-uniform memory access costs. The cost of memory access depends on whether the processor that hosts the memory controller managing the memory address is the same as the processor accessing the memory address. This metric displays the percentage of observed memory accesses that conflicted with another memory access to the same variable at the NUMA level.

3.2.5 Reuse factors for level 1, 2 and 3 data caches:

From the observed reuse distance and a probabilistic model of set-associative caches, MACPO estimates whether a given memory access would be served out of L1 or would overflow the size of the cache, resulting in the memory access being served out of a higher (L2 or possibly L3) level of cache. This analysis permits MACPO to calculate the multi-core reuse factor, which is an estimate of the occurrence of a given address being reused within a specific cache level.

4. Interpreting MACPO metrics

4.1 Cycles per access

The cycles per access metric provides an overview of the performance of memory accesses to a specific variable. It makes it possible to identify whether a particular variable is suffering from memory performance bottleneck problems.

4.2 Dominant stride values and percentages

Programs that have unit strides or small regular strides generally execute faster than programs that have long or irregular access strides. Unit strides increase reuse in cache, improve the efficiency of prefetchers and reduce the need to refill the Translation Lookaside Buffer.

4.3 Cache conflicts

Cache conflicts indicate thrashing between caches. By padding the data structures with additional (dummy) bytes, each thread can be made to access a different cache line, effectively removing cache conflicts.

4.4 NUMA conflicts

Most operating systems implement a first-touch policy by which the memory controller associated with the processor that first accesses the memory, "owns" the memory address. Memory accesses for the same address by a different processor result in NUMA conflicts. NUMA conflicts typically arise when one thread initializes memory which is then accessed by different threads. NUMA conflicts can be avoided if each processor initializes it's own memory.

4.5 Reuse factors for level 1, 2 and 3 data caches

A low reuse factor indicates that a line is frequently evicted from the cache. The reuse factor can be improved by reducing reuse distance of memory accesses.