PufferFish: NUMA-Aware Work-stealing Library using Elastic Tasks

Vivek Kumar

Department of Computer Science and Engineering, IIIT Delhi, India
vivekk@iiitd.ac.in

Abstract. Due to the challenges in providing adequate memory access to many cores on a single processor, Multi-Die and Multi-Socket based multicore systems are becoming mainstream. These systems offer cache-coherent Non-Uniform Memory Access (NUMA) across several memory banks and cache hierarchy to increase memory capacity and bandwidth. Random work-stealing is a widely used technique for dynamic load balancing of tasks on multicore processors. However, it scales poorly on such NUMA systems for memory-bound applications due to cache misses and remote memory access latency. Hierarchical Place Tree (HPT) [3] is a popular approach for improving the locality of a task-based parallel programming model, albeit it requires the programmer to map the dynamically unfolding tasks over a NUMA system evenly. Specifying data-affinity hints provides a more natural way to map the tasks than HPT. Still, a scalable work-stealing implementation for the same is mostly unexplored for modern NUMA systems.
This software includes an implementation of *PufferFish* [1], a new **async–finish** parallel programming model and work-stealing runtime for NUMA systems that provide a close coupling of the data-affinity hints provided for an asynchronous task with the HPTs in Habanero C/C++ library (`HClib`). `PufferFish` introduces *Hierarchical Elastic Tasks (HET)* that improves the locality by shrinking itself to run on a single worker inside a `place` or puffing up across multiple workers depending on the work imbalance at a particular `place` in an HPT. We also include the source code of the different benchmarks presented in the paper.

1 Getting Started

`PufferFish` was evaluated on an AMD EPYC 7551 32-Core processor running at minimum and maximum frequencies of 1.2 GHz and 2.0 GHz, respectively, and with a total of 64GB of RAM. The operating system was Ubuntu 18.04.3 LTS. The GCC compiler version was 7.5.0. We used the CilkPlus version shipped with this GCC compiler.

1.1 Software Installation

1. Directory setup:

```
cd pufferfish
#Environment setup inside each bash shell
vim ./setup-env.sh
#Change the variable ``PUFFERFISH'' such that it
#holds the absolute path to directory ``pufferfish''
#Save and quit
source ./setup-env.sh
```

2. Download and patch HClib runtime

```
cd pufferfish/runtime
#Script will patch on HClib GitHub version ``ab310a0''
./patch-hclib.sh
```

3. Cilk runtime installation

```
sudo apt-get install -y libcilkrts5
```

2 Step-by-Step Instructions on how to use **PufferFish**

2.1 Build runtimes and compile benchmarks

HClib variants can be built with/without using LIKWID. When built with LIKWID, there will be performance counter measurements overhead [2].

```
#Build HClib runtime variants without the support for LIKWID
cd pufferfish/runtime
#Build three different runtimes based on HClib:
# a) pufferFish,
# b) defaultRand (default HClib with random work-stealing),
# c) hptDA
./build-hclib.sh

#Build HClib runtime variants with the support for LIKWID
cd pufferfish/runtime
./clean.sh
#Build three different runtimes based on HClib:
# a) pufferFish,
# b) defaultRand (default HClib with random work-stealing),
# c) hptDA
./build-likwid-hclib.sh
```

2.2 Speedup over sequential

```
#Launch experiments that would execute Sequential
#variants of the 7 benchmarks for 10 times. The output
#log files will be stores inside the following
```

```
#directory: ``pufferfish/logs''. Note that this is
#a long running experiment.
cd pufferfish/benchmarks
./experiment-seq.sh #this also invokes appropriate makefiles
#Launch experiments that would execute each benchmark
#10 times with six different implementations: (a) HClib (FT)
#(b) HClib (IL), (c) PufferFish, (d) HCLIB (HPT_DA),
#(e) CilkPlus (FT), and (f) CilkPlus (IL)
#The output log files will be stores inside the following
#directory: ``pufferfish/logs''. Note that this is
#a long running experiment.
./experiment.sh #this also invokes appropriate makefiles
#Performance Analysis
cd pufferfish/experiments
./speedup-over-Sequential.pl
```

2.3 Hardware Performance Counter Experiment

HClib variants used in these experiments are built using LIKWID as mentioned in Section 2.1 above.

```
#Launch experiments that would execute each benchmark
#10 times with four different implementations: (a) HClib (FT)
#(b) HClib (IL), (c) PufferFish, and (d) HCLIB (HPT_DA),
#The output log files will be stores inside the following
#directory: ``pufferfish/logs-pcm''. Note that this is
#a long running experiment.
./experiment-pcm.sh #this also invokes appropriate makefiles
cd pufferfish/experiments
./relative-performanceCounters.pl
#Memory Access Pattern in Benchmarks
cd pufferfish/experiments
./memoryAccessPattern.pl
```

3 Manual execution of benchmarks using PufferFish

Follow the instructions below for manually executing a benchmark using `PufferFish`.

1. Patch HClib by following the instructions mentioned in Section 1.1.
2. Prepare an HPT file for your NUMA machine by using the HPT files of AMD EPYC 7551 processor as a template:
   - One NUMA node: "$PUFFERFISH/runtime/hclib/hpt/hpt-hippo-1numa.xml".
   - Two NUMA node: "$PUFFERFISH/runtime/hclib/hpt/hpt-hippo-2numa.xml".

- Three NUMA node: "$PUFFERFISH/runtime/hclib/hpt/hpt-hippo-3numa.xml".
- Four NUMA node: "$PUFFERFISH/runtime/hclib/hpt/hpt-hippo-4numa.xml".

Note that these HPT files are hardcoded in the experimental scripts inside the directory "$PUFFERFISH/benchmarks" for AMD EPYC 7551 processor.

3. Prepare the Makefile for compiling your benchmark by using the file "$PUFFERFISH/benchmarks/makefile.pufferFish" as a template.
4. Build PufferFish:

```
cd /Path to HClib directory/hclib
HCLIB_FLAGS=" --enable-pufferFish --enable-likwid ./install.sh
Or
HCLIB_FLAGS=" --enable-pufferFish ./install.sh
```

5. Compile your benchmark:

```
cd /Path to benchmark/
source /Path to HClib directory/hclib-install/hclib_setup_env.sh
make -f makefile.your
```

6. Execute your benchmark as follows:

```
source /Path to HClib directory/hclib-install/hclib_setup_env.sh
export HCLIB_HPT_FILE=/HPT directory path/hpt.xml
export HCLIB_BIND_THREADS=1
export HCLIB_STATS=1
./executable <command line arguments to benchmark>
```

7. Execution time of the compute kernel will be shown by the runtime as "time.kernel". Total execution time of the benchmark is shown as "Total time". Total number of **`async_hinted`** created in the benchmark is shown as "totalNumaPush". Total number of **`async_hinted`** actually pushed on workers deque (due to elastic tasks) is shown as "totalPush".

References

1. Kumar, V.: PufferFish: NUMA-aware work-stealing library using elastic tasks. In: 2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC). IEEE (2020), to appear
2. Röhl, T., Treibig, J., Hager, G., Wellein, G.: Overhead analysis of performance counter measurements. In: 2014 43rd International Conference on Parallel Processing Workshops. pp. 176–185 (2014). https://doi.org/10.1109/ICPPW.2014.34
3. Yan, Y., Zhao, J., Guo, Y., Sarkar, V.: Hierarchical place trees: A portable abstraction for task parallelism and data movement. In: LCPC '10. pp. 172–187 (2010). https://doi.org/10.1007/978-3-642-13374-9_12