GPUs in StarPU

Ronak Buch

9 October 2018

StarPU Overview

- StarPU is a task based programming model
- Programs are written as:
  - *Codelets* - kernel, potentially with multiple implementations
  - *Tasks* - *codelets* applied to a data set
- *Tasks* are asynchronous and specify data dependencies
- Scheduler generally infers task dependencies via data

GPU Support

- Supports CUDA and OpenCL
- Manages data transfers to/from accelerators
- Offers abstraction layer that maps to correct implementation (e.g. `starpu_malloc` will use CUDA APIs)

Code Samples - Overall

```
//specifies codelet
struct starpu_codelet cl =
{
.cpu_funcs = { cpu_func },
.nbuffers = 0
};

void cpu_func(void* buffers[], void *cl_arg) {
  struct params *params = cl_arg;
  printf("Hello world - %i, %f)\n", params->i, params->f);
}

int main(int argc, char** argv) {
  starpu_init(NULL);
  struct starpu_task
  task = starpu_task_create();
  task->cl = &cl;
  struct params params = { 1, 2.0f };
  task->cl_arg = &params;
  task->cl_arg_size = sizeof(params);
  task->synchronous = 1;
  starpu_task_submit(task);
  starpu_shutdown();
}
```

Code Samples - CUDA Codelet

```
extern void scal_cuda_func(void *buffers[], void *_args);

static struct starpu_codelet cl = {
  .where = STARPU_CPU | STARPU_CUDA,
  .cpu_funcs = { scal_cpu_func },
  .cpu_funcs_name = { "scal_cpu_func" },
  #ifdef STARPU_USE_CUDA
  .cuda_funcs = { scal_cuda_func },
  #endif
  .nbuffers = 1,
  .modes = { STARPU_RW }
};
```

Code Samples - CUDA Kernel

```
#include <starpu.h>

static __global__ void vm_cuda(unsigned n, float* val, float
    factor) {
  unsigned i =  blockIdx.x * blockDim.x + threadIdx.x;
  if (i < n)
    val[i] *= factor;
}

extern "C" void scal_cuda_func(void *buffers[], void *_args) {
  float *factor = (float *)_args;
  unsigned n = STARPU_VECTOR_GET_NX(buffers[0]);
  float *val = (float*)STARPU_VECTOR_GET_PTR(buffers[0]);
  unsigned threads_in_block = 64;
  unsigned nblocks = (n + threads_in_block-1) / threads_in_block;
  vm_cuda<<<nblocks,threads_in_block, 0,
      starpu_cuda_get_local_stream()>>>(n, val, *factor);
  cudaStreamSynchronize(starpu_cuda_get_local_stream());
}
```

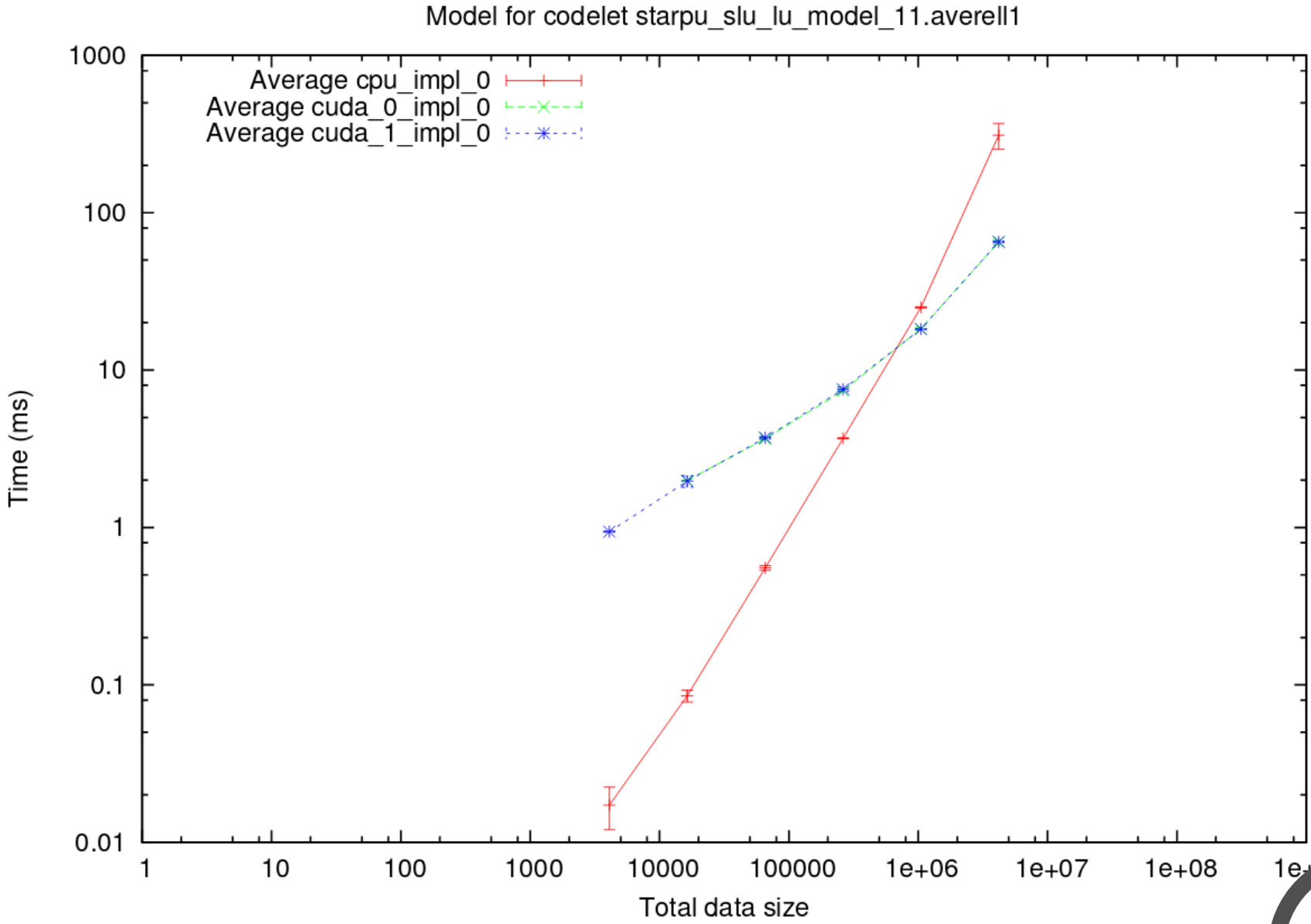
Performance Models

- Can automatically collect runtime performance information
- Decisions can be based on exact data sizes or regression
- Provide `.model` parameter to codelet
- Only some schedulers use this information
- Saved to file and can be reused across runs

Performance Models

```
$ starpu_perfmodel_display -s starpu_slu_lu_model_22
performance model for cpu
# hash      size        mean           dev            n
57618ab0    19660800    2.851069e+05   1.829369e+04   109
performance model for cuda_0
# hash      size        mean           dev            n
57618ab0    19660800    1.164144e+04   1.556094e+01   315
performance model for cuda_1
# hash      size        mean           dev            n
57618ab0    19660800    1.164271e+04   1.330628e+01   360
performance model for cuda_2
# hash      size        mean           dev            n
57618ab0    19660800    1.166730e+04   3.390395e+02   456
```

Performance Models



Misc.

- Dedicates one CPU core per GPU
- Provides streams for users to use
- Supports converting data layouts (Multiformat)
- Supports energy based scheduling, but not online for CUDA (build offline model)
- Can specify specific worker (e.g. GPU0)
- Offers abstraction layer that maps to correct implementation (e.g. `starpu_malloc` will use CUDA APIs)