

PRUDA: Time and Space Predictible Programming for NVIDIA GPUs using CUDA

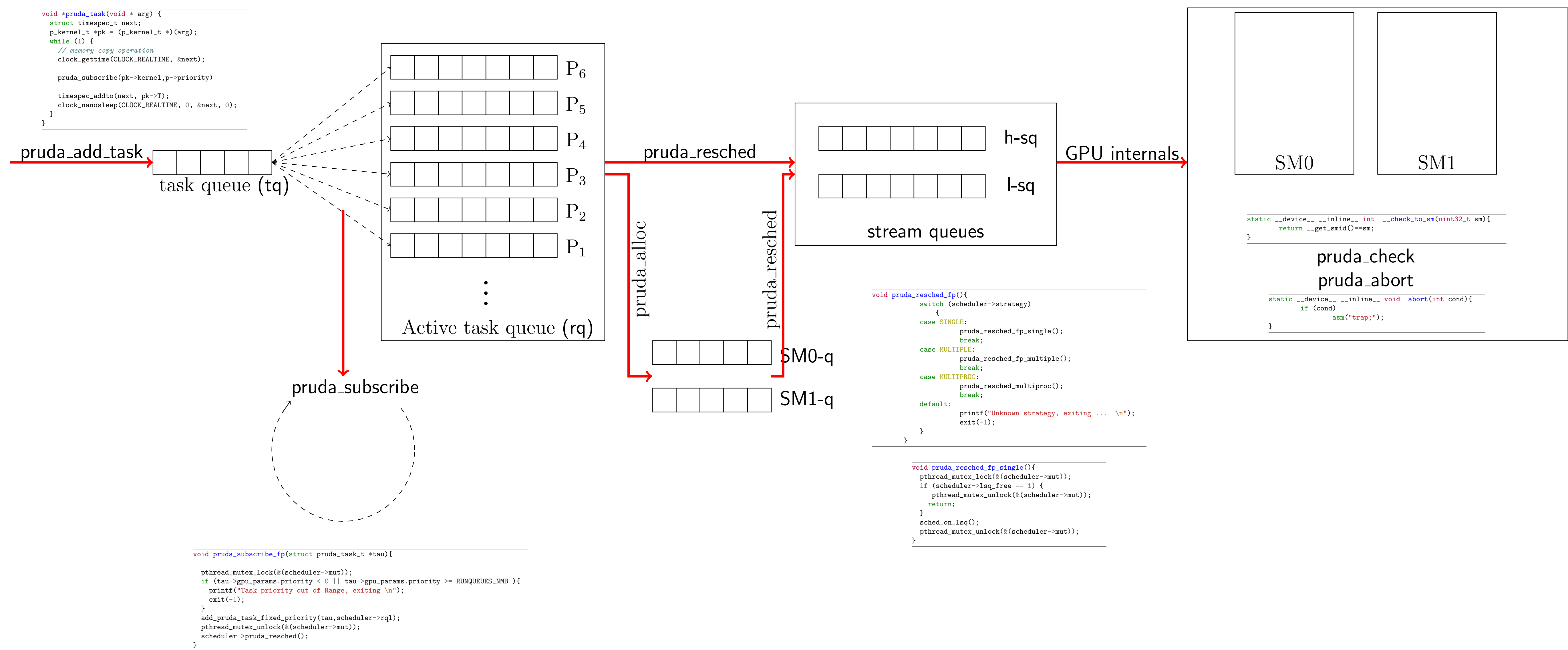


Reyyan Tekin, Houssam-Eddine Zahaf, Giuseppe Lipari

Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL, Lille, France

{firstname.familyname}@univ-lille.fr

Overview of PRUDA Architecture



Temporal and Spatial control : Strategies

	Single-stream strategy	Multiple stream	SMs as cores strategy
Advantages	<ul style="list-style-type: none">Simple, easy to implementImplicit stream synchronization	<ul style="list-style-type: none">Allow concurrent kernel execution on GPUsSupports block-level preemptions	<ul style="list-style-type: none">Allow parallel execution in the GPUProvides temporal and spatial execution control
Drawbacks	<ul style="list-style-type: none">Use the GPU as a single coreResources may be wastedSupport Only non-preemptive algorithms	<ul style="list-style-type: none">Use the GPU as a single coreResources may be wastedOnly block-boundaries preemptions.	<ul style="list-style-type: none">Complex to implementUnusual schedulability analysis (new)

Examples and performances

```
#define N 8

__global__ void add(int *a, int *b, int *c) {
    int tid = blockIdx.x * blockDim.x + threadIdx.x;

    while (tid < N) {
        c[tid] = a[tid] + b[tid];
        tid += blockDim.x;
    }
}

int main() {
    int *dev_a, *dev_b, *dev_c;

    //Memory allocation and initializing

    //Kernel initializing
    init_kernel_listing();
    create_kernel(std::get<0>(get_listing()), add, gridSize, blockSize, dev_a, dev_b, dev_c);

    //Periodic task parameters
    struct gpu_sched_param ga;
    ga.period_us = 5000;
    ga.deadline_us = 5000;
    ga.priority = 3;

    //Periodic task creation
    struct pruda_task_t * p_task_a = create_pruda_task(0, ga, gridSize, blockSize);

    //Using Fixed Priority algorithm with the Single-stream strategy
    init_scheduler(SINGLE, FP);
    add_pruda_task(p_task_a);

    //One CPU thread per CUDA kernel (see pruda_task() function example above)
    create_cpu_threads();

    //CPU operations

    //Memory free

    return 0;
}
```

- No modification required for CUDA programming style.
- Simple to use, and to configure.
- Generic : Kernels with different signatures are handled (via variadic templates)

Source code available on : <https://gitlab.cristal.univ-lille.fr/ptask/rtgpgpu>

