JLESC 2020

Design of Runtime System for Task-based FPGA Programming

Programming Environment Research Team
Jinpil Lee

Agenda

- Task offloading to FPGA using task
- Batch generation in runtime

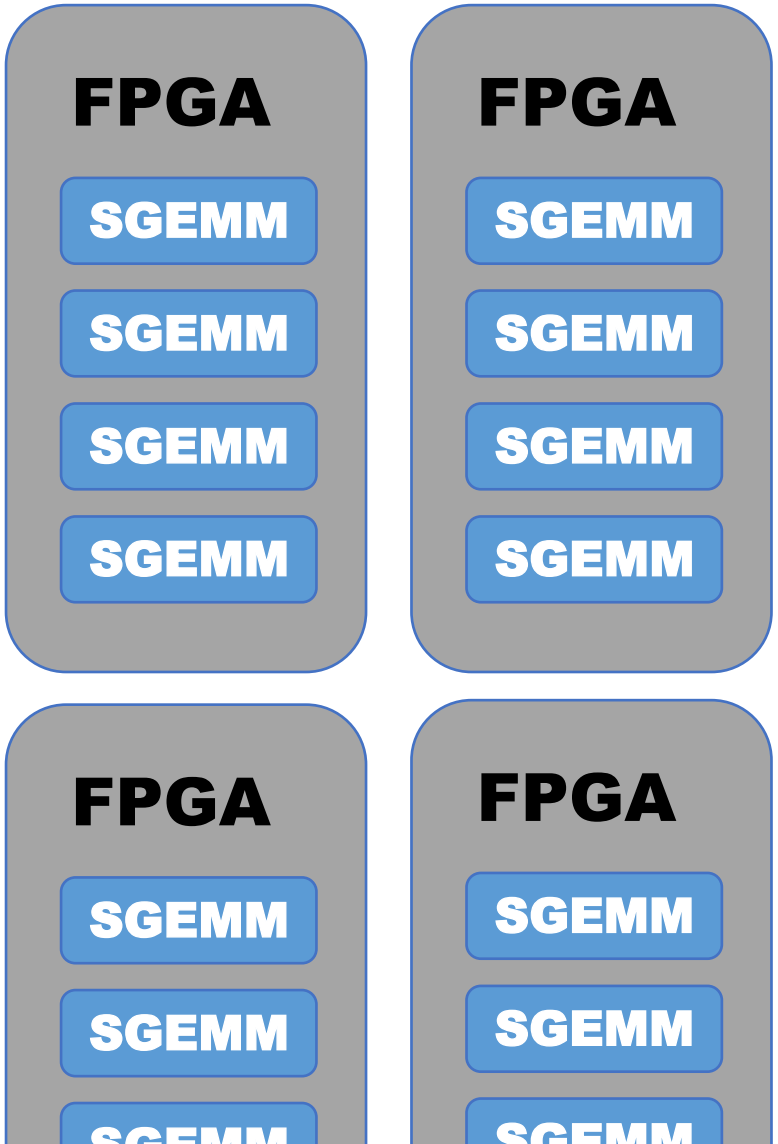
Background

- Writing computing kernels for FPGAs
  - OpenCL/OpenMP/OpenACC/DPC++/DSL
- How to control the generated kernel?
  - OpenCL event
  - OpenACC kernel
  - OpenMP target

  generates FPGA kernels
  - XcalableMP task

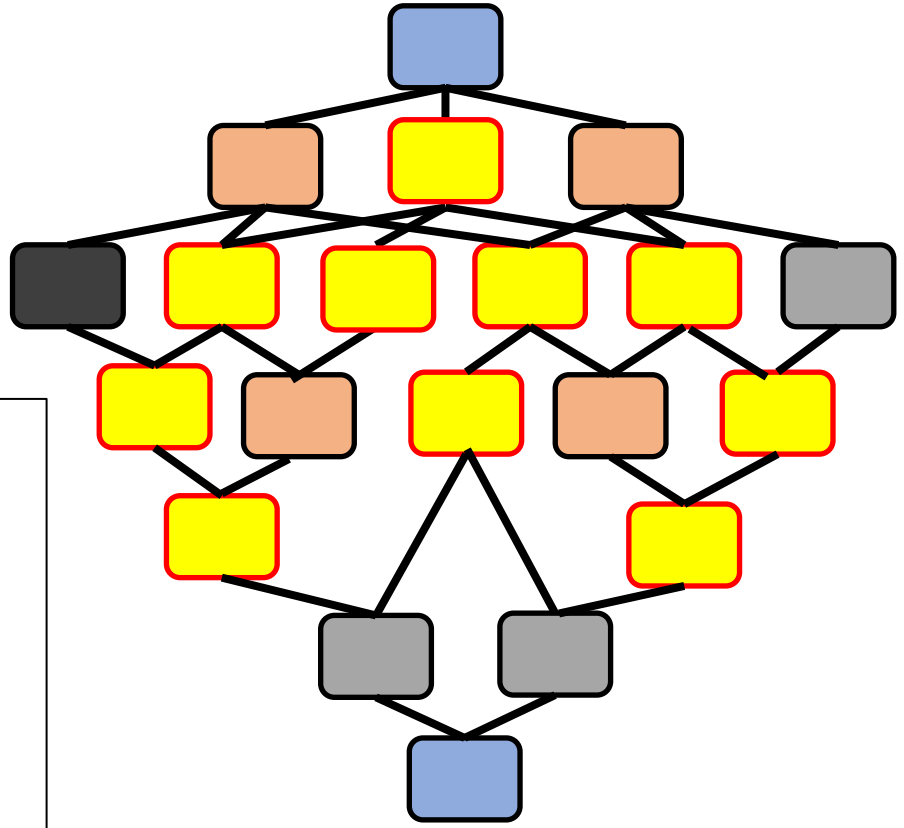
  assumption

  1. optimized libraries using FPGAs
     (e.g. BLAS for FPGA)
  2. APIs are called by CPUs
  3. inner-node parallelism
     (inter-node work distribution is the future work)



Motivated Example

- In some OpenMP task program, tasks like yellow part will appear
  - dominant computation time
  - can be executed in parallel by OpenMP task
- Host can offload these tasks asynchronously
  - overlap between CPU and FPGA



```
#pragma omp task depend …
{
   …
   calc_on_fpga(event);
   do {
#pragma omp taskyield
      status = checkStatus(event);
   } while (status != done);
   …
}
```

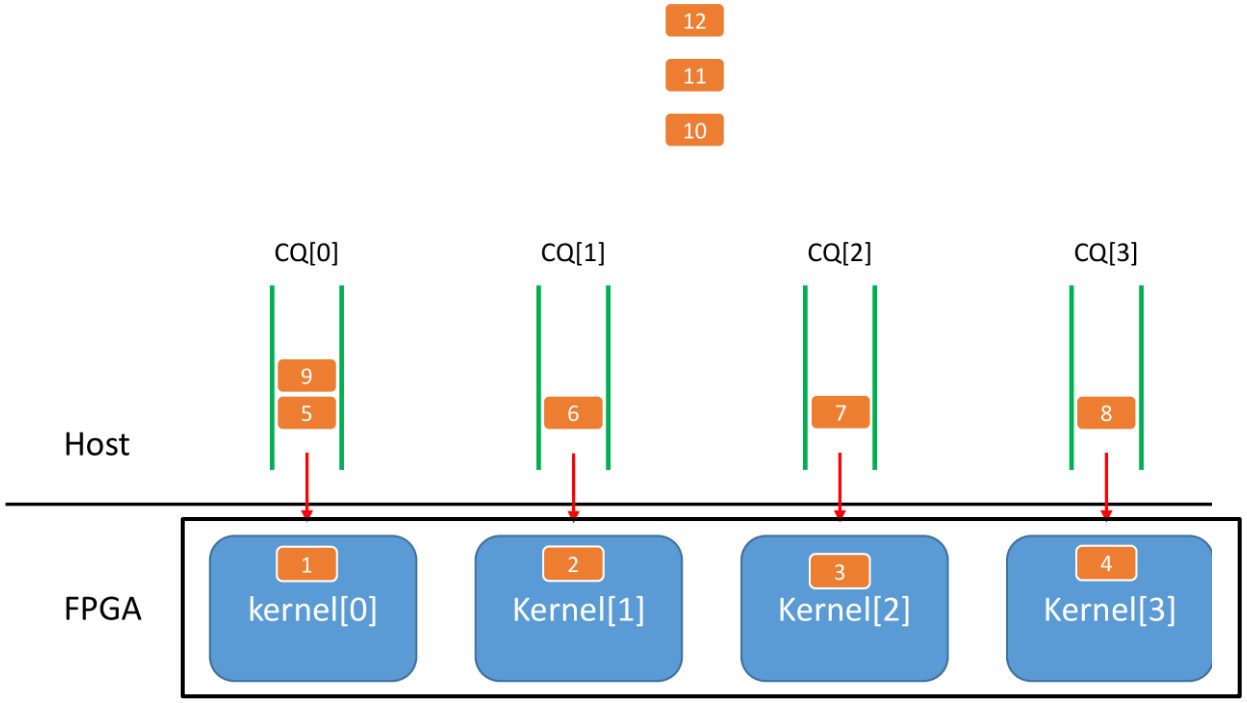
```
#pragma xmp task depend async
  on dev(FPGA) …
{
  calc(event);
}

#pragma xmp variant(FPGA:calc)
{
  calc_on_fpga(event);
}
```

Implementation

- Prepared independent queue for each FPGA instance
  - parallel execution
  - cl_command_queue… like CUDA stream object
- Asynchronous task execution
  - Argobots

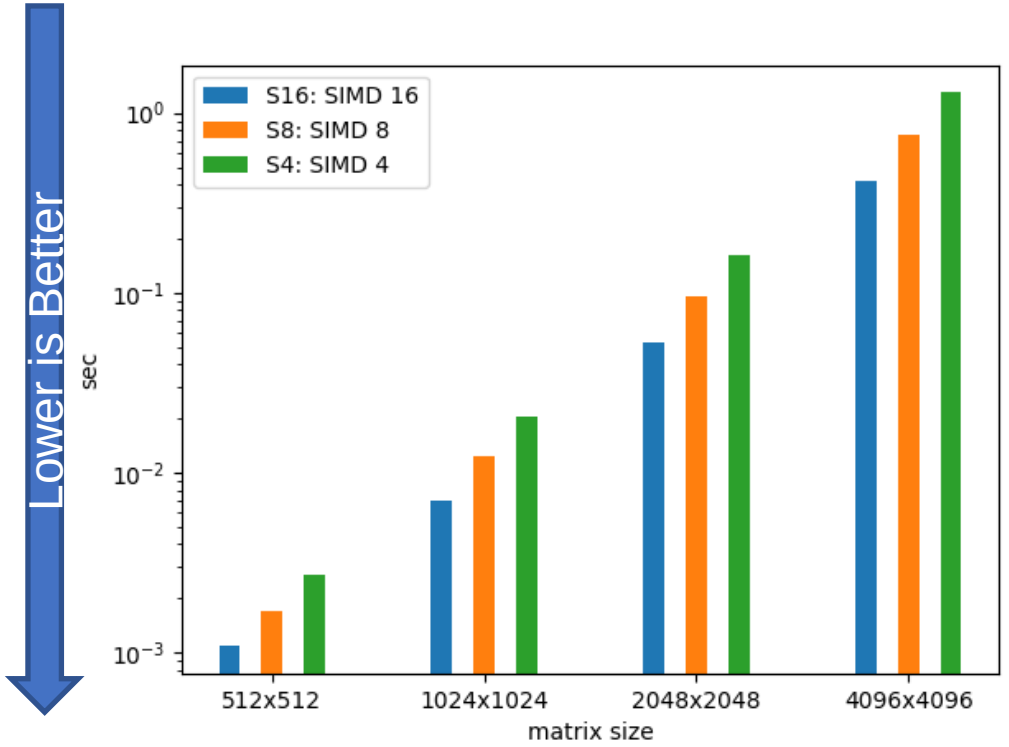
Hardware configuration	
CPU	Intel Xeon E5-2660 v4 x 2
Host DRAM	DDR4-2400 16GB x 4
FPGA Board	BittWare A10PL4 (Intel Arria10 GX1150) PCIe Gen3 x8
FPGA DRAM	DDR4-2133 4GB x 2

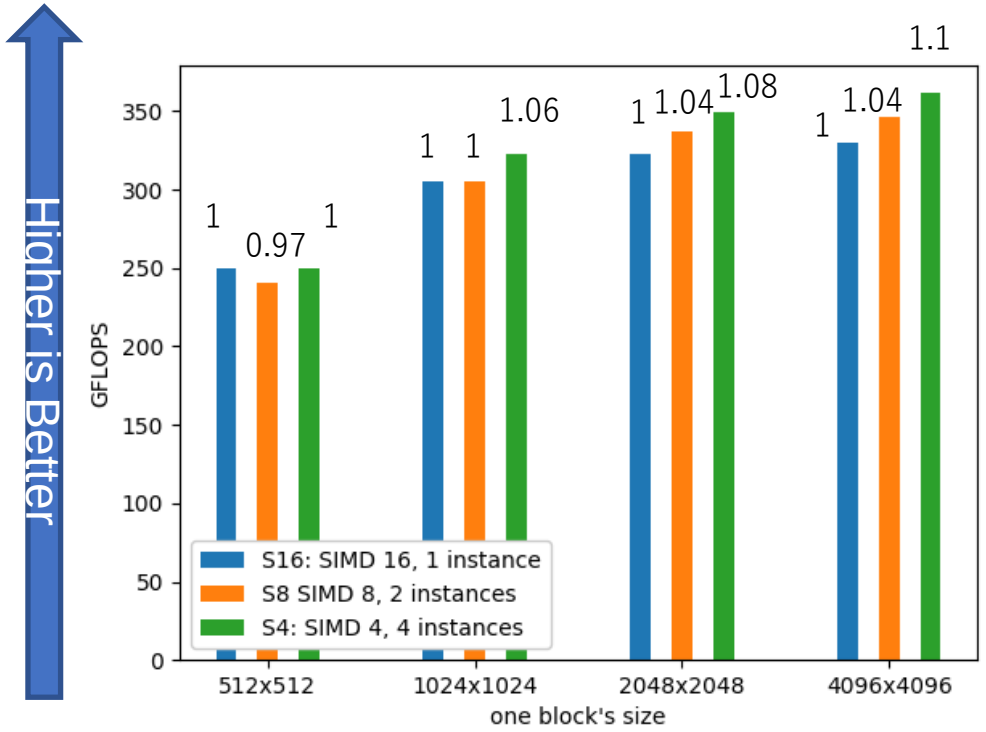
Software configuration	
OS	CentOS 7.3 x64
Host Compiler	Intel C Compiler 18.1
Thread library	Argobots 1.0b1
FPGA compiler	Intel FPGA SDK for OpenCL 17.12.304



SIMD vs Instance

	SIMD width	num. of instances	Fmax (MHz)	DSPs (%)	BRAM (%)
S16	16	1	164.71(1.0)	69	60
S8	8	2	179.59(1.09)	69	69
S4	4	4	213.94(1.29)	70	90





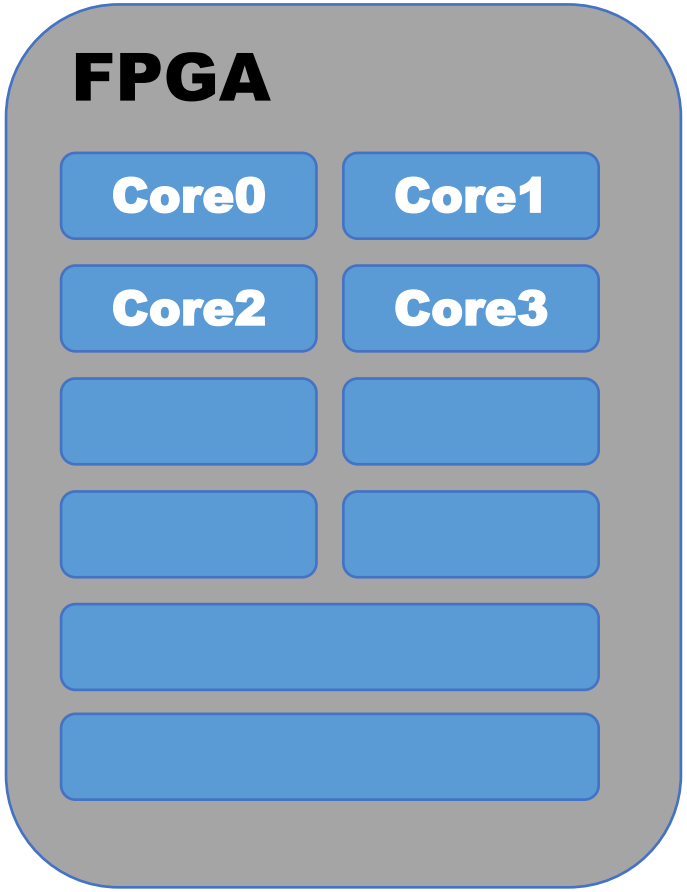
Parallelism on FPGAs

Multiple Instances with

- Independent task queue
  - ▶ like multi-core CPUs
  - ▶ can use XMP/OMP task

- Single task queue
  - ▶ like GPUs
  - ▶ internal hardware scheduler
  - ▶ single task will use the whole device

→ requires parallelism **inside a task**



Batched Kernel API

- batch: gathering multiple calculation kernels into a single task
- implementation depends on the target hardware
  - CPU/GPU implementation exploits its parallelism
  - assign kernels onto computing cores
- needs code modification

BLAS API

```
void calc_dgemms(const int num_kernels,
    const int n, double **A, double **B,
    double **C, double *DMONE,
    double *DONE) {
  for (int i = 0; i < num_kernels; i++) {
    cublasDgemm(handle, CUBLAS_OP_N,
        CUBLAS_OP_N, n, n, n,
        &DMONE, A[i], n, B[i], n,
        &DONE, C[i], n);
    cudaDeviceSynchronize();
  }
}
```

Batched BLAS API

```
void calc_dgemms_batched(const int num_kernels,
    const int n, double **A, double **B, double **C,
    double *DMONE, double *DONE) {
  cublasDgemmBatched(handle, CUBLAS_OP_N,
      CUBLAS_OP_N, n, n, n, &DMONE, A, n, B, n,
      &DONE, C, n, num_kernels);
  cudaDeviceSynchronize();
}
```

Code Translation from OMP

- static vs dynamic
- static
  - no runtime overhead
  - covers easy cases
- **dynamic**
  - simple
  - runtime overhead
  - irregular patterns

```
for (int i = k+1; i < num_tiles; i++) {
  for (int j = k+1; j < i; j++) {
#pragma omp task depend(in:A[k][i],A[k][j]) depend(out:A[j][i])
    dgemm(A[k][i], A[k][j], A[j][i], tile_size, tile_size);
  }
#pragma omp task depend(in:A[k][i]) depend(out:A[i][i])
  dsyrk(A[k][i], A[i][i], tile_size, tile_size);
}
```

```
for (int i = k+1; i < num_tiles; i++) {
#pragma omp task depend(in:A[k][i],A[k][K+1:i]) ¥¥
                                depend(out:A[K+1:i][i])
  dgemm_batch(A_ptr_array, ...);
#pragma omp task depend(in:A[k][i]) depend(out:A[i][i])
  dsyrk(A[k][i], A[i][i], tile_size, tile_size);
}
```

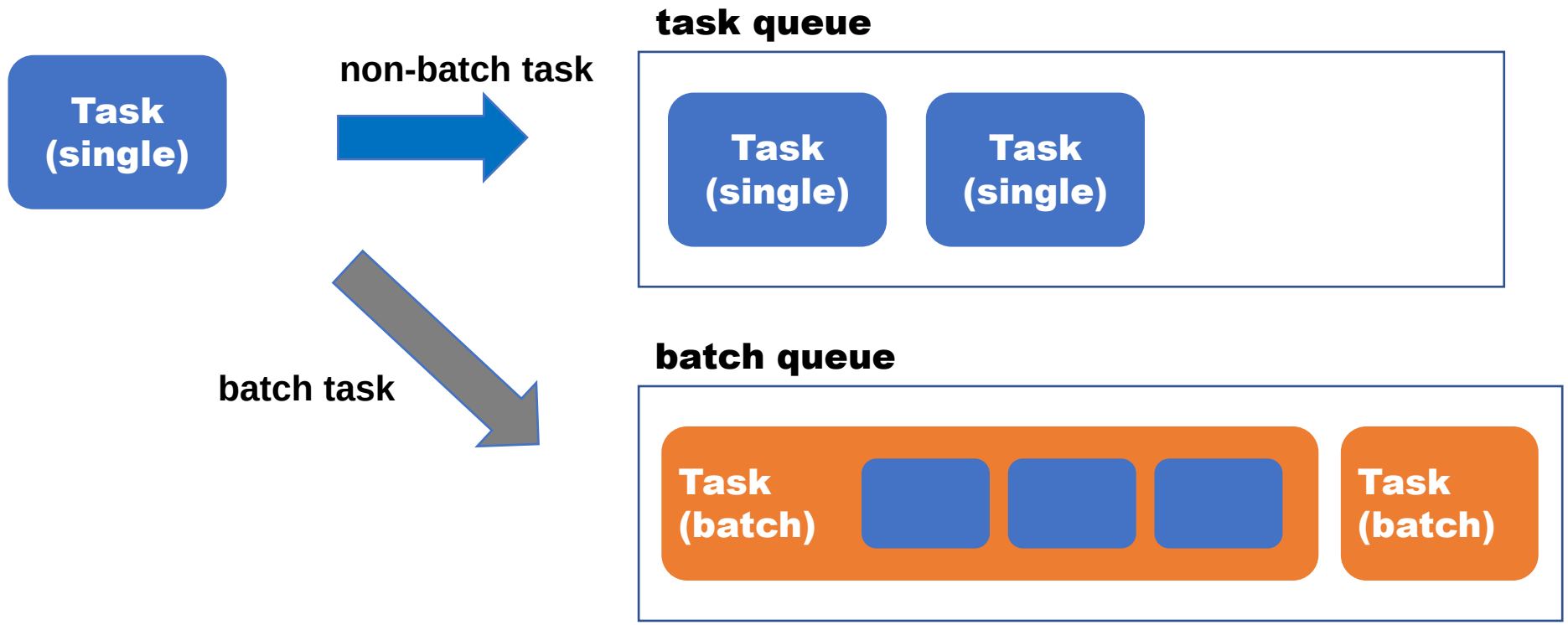
Translated Code

- Front-end generates unique batch IDs
  - function name, argument pointer/value
- 0: non-batch
- task runtime
  - task_create()
  - task_waitall()

```
void cholesky_decomposition(double* A[nt][nt], ...) {
  for (int k = 0; k < nt; k++) {
    void *args[3] = {A[k][k], &tile_size, &tile_size};
    void *out_data[1] = {A[k][k]};
    task_create(0, potrf, 3, args, 0, NULL, 1, out_data);
    for (int i = k + 1; i < nt; i++) {
      void *args[4] = {A[k][k], A[k][i], &tile_size, &tile_size};
      void *in_data[1] = {A[k][k]};
      void *out_data[1] = {A[k][i]};
      task_create(0, trsm, 4, args, 1, in_data, 1, out_data);
    }
    for (int i = k + 1; i < nt; i++) {
      for (int j = k + 1; j < i; j++) {
        void *args[5] = {A[k][i], A[k][j], A[j][i], &tile_size, &tile_size};
        void *in_data[2] = {A[k][i], A[k][j]};
        void *out_data[1] = {A[j][i]};
        task_create(1, gemm, 5, args, 2, in_data, 1, out_data);
      }
```

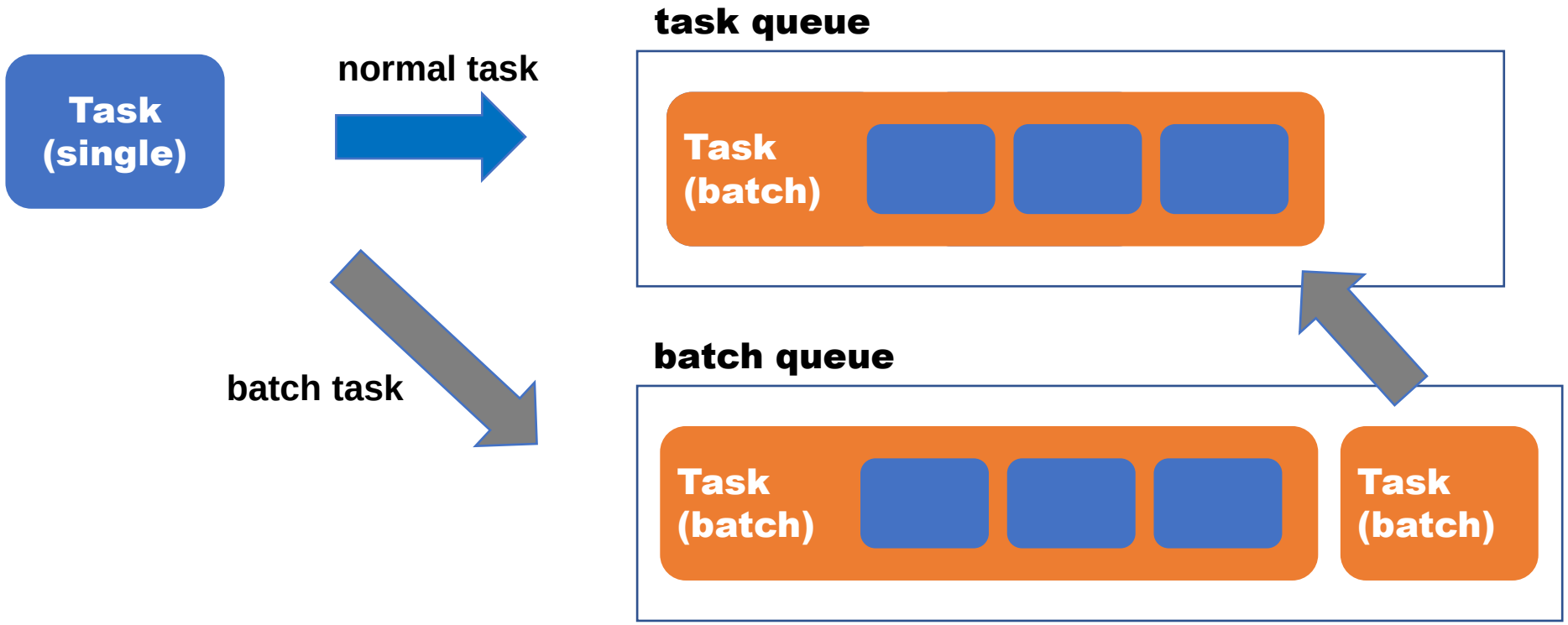
Task Runtime

- non-batch tasks use normal OpenMP runtime
- batch task
  - task with batch ID
  - put into batch queue
  - merged when the same kind of task exists



Task Runtime (cont'd)

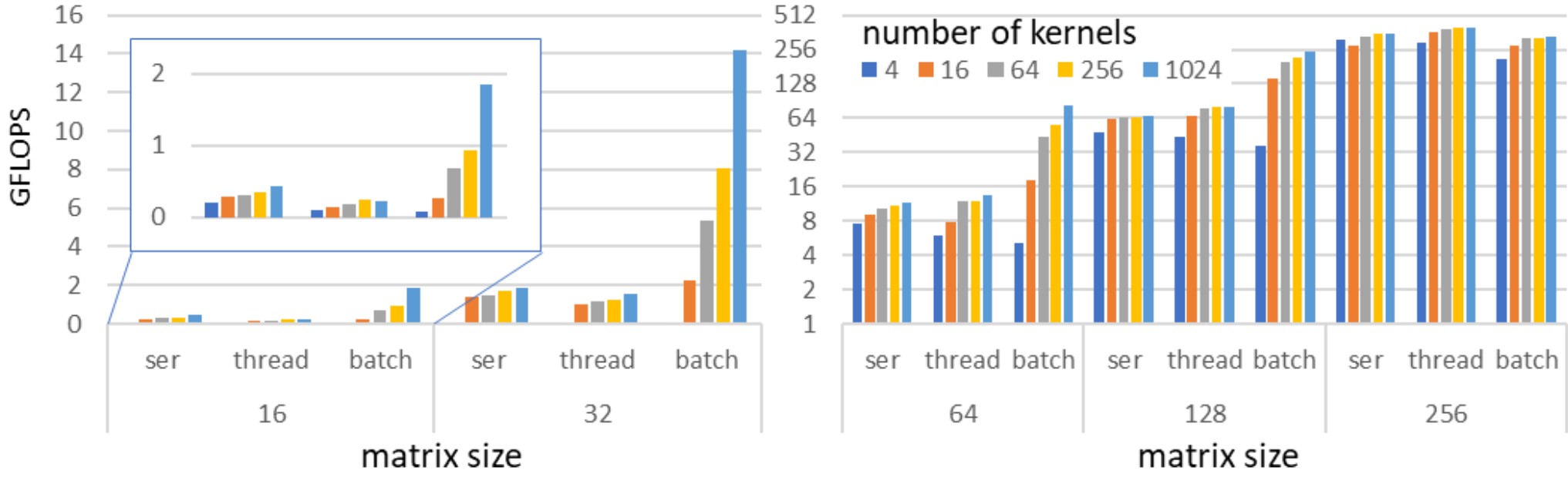
- task scheduling algorithm uses the task queue
  - (batch) task is not ready for task scheduling when in the batch queue
- batch tasks are moved to the task queue
  - when the task queue is empty (ready for scheduling)
  - delaying scheduling to gather more kernels into a batch



Evaluation

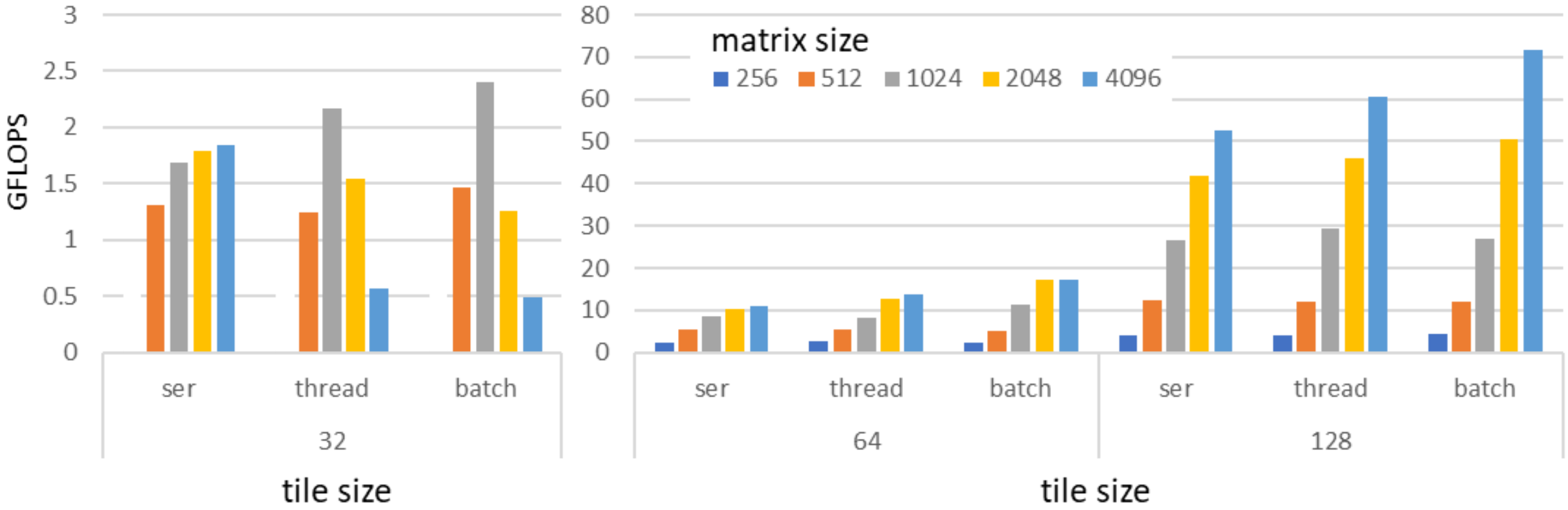
- DGEMM Loop
  simple、shows the ideal performance
- Blocked Cholesky Decomposition
  to see the runtime overhead
- single thread, all serialized

Item	Name
CPU	Intel (R) Xeon (R) CPU E5-2680 8 cores (2 sockets), 2.70 GHz
	Intel Compiler 18.0.3
GPU	NVIDIA Tesla K20
	CUDA 9.1 with cuBLAS
Memory	DDR4 64 GB



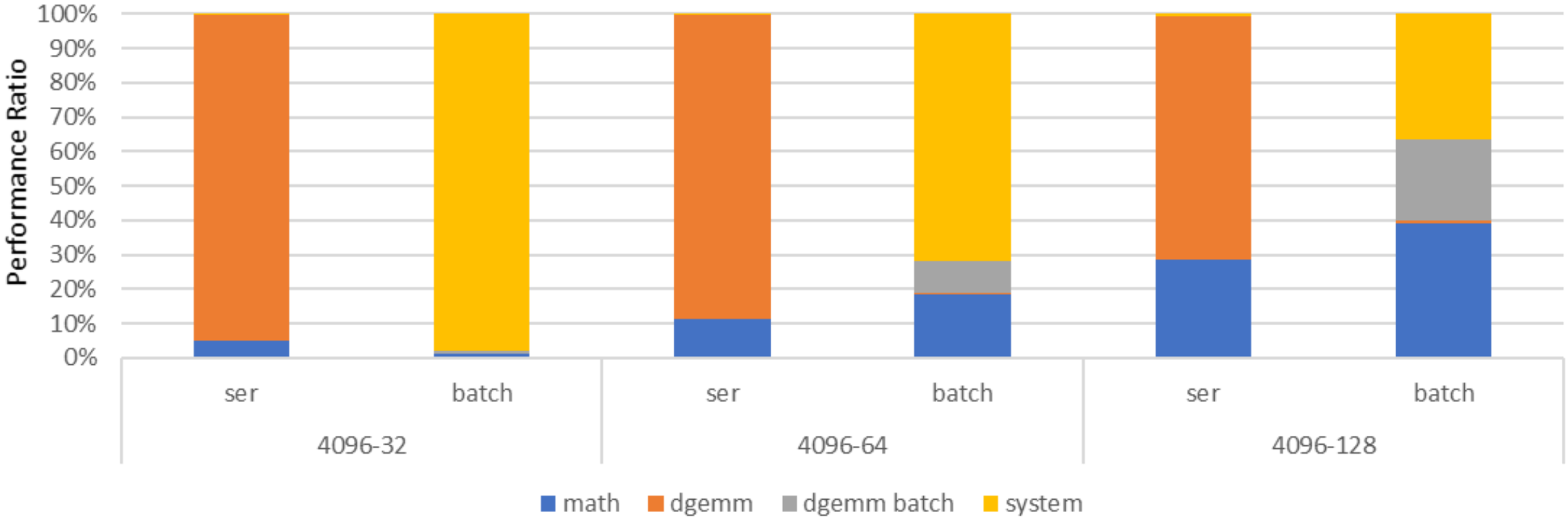
Result: Cholesky Decomposition

- block (tile) size: 128, # Blocks: 4096 → 36% perf improvement
- little improvement with small matrices
- not consistent with the previous result ...



Performance Breakdown

- system: overall runtime overhead (batch creation, scheduling)
- huge performance improvement & overhead with small matrices
- trade off between batch efficiency and task overhead?



Future Plan

- Work distribution among FPGA nodes
  - data distribution
  - scheduling tasks among nodes by dependency

- Remote Procedure Call from Fugaku to ESSPER
  - by XcalableMP?