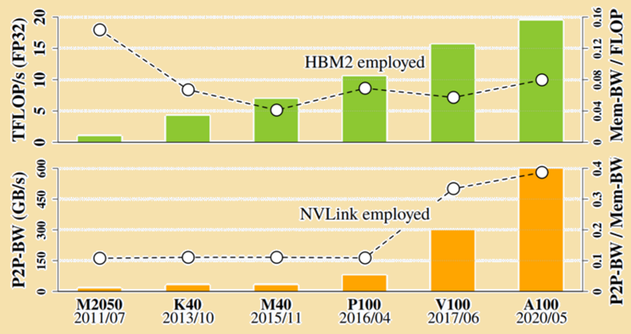
Wrapping up Existing OpenACC Compilers for Runtime Extension

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

INTRODUCTION



GPUs' Performance Change

- Higher computer performance,
  Lower Bandwidth/FLOP
- ***Increased***
  P2P-Bandwdith/Mem-Bandwidth [1]

OpenACC with Multi-GPU

- Device code with directives [2]
- ***In-situ kernel declarations bring complexities to code maintenance***

```
Config  for (int d = 0; d < NUM_DEVICES; d++) {
            acc_set_device_num(d , 0);
            int length = N / NUM_DEVICES;
            int init = length * d;
            int until = length * (d + 1);
Kernel      #pragma acc parallel loop independent async(d)
            for (i = init; i < until; i++) {
                /* int... */
            }
Comm        for (int d2 = 0; d2 < NUM_DEVICES; d2++) {
                cudaMemcpyAsync( … );
            }
        }
```

01 JACC: Runtime-Extended OpenACC

- **JACC facilitates dynamic extension by providing extendable routines**

Input Code (C/Fortran)

```
#pragma acc data copyout(x[0:N])
#pragma acc parallel loop
for(int i=0; i<N; i++)
  x[i]=y[i] * y[i];
```

① Routine Use

JACC Code

```
/* Entry of #pragma acc data */
jacc_create(x, N * sizeof (float));

/* #pragma acc parallel loop */
jacc_kernel_push(
  "#pragma acc parallel present (x, y)\n"
  "#pragma acc loop\n"
  "for(int i=0; i<N; i++) /* ... */",
  /* args */ , /* flags */ );

/* Exit of #pragma acc data */
jacc_copyout(x, N * sizeof(float));
```

② Exec

Dynamic Code

```
void kernel0(float *x, float *y, size_t N) {
#pragma acc parallel present(x, y)
#pragma acc loop
for(int i=0; i<N; i++)
    x[i]=y[i] * y[i];
}
```

④ Compile (PGI/GCC)
③ Generate Code
Link
Caching Binary
Execution

JACC Runtime

Data Management	Kernel Execution
- Array Mapping	- Code Gen/Compile
- CPU-to/from-Device Communication	- Parameter Calculation
	- Extended Execution

02 ASYNC & KERNEL SPECIALIZATION

Asynchronous Execution

- Data-tracking with red-black tree
- Read/Write check

```
void kernel0(…, int async) {
#pragma acc parallel … async(async)
    …
}
```

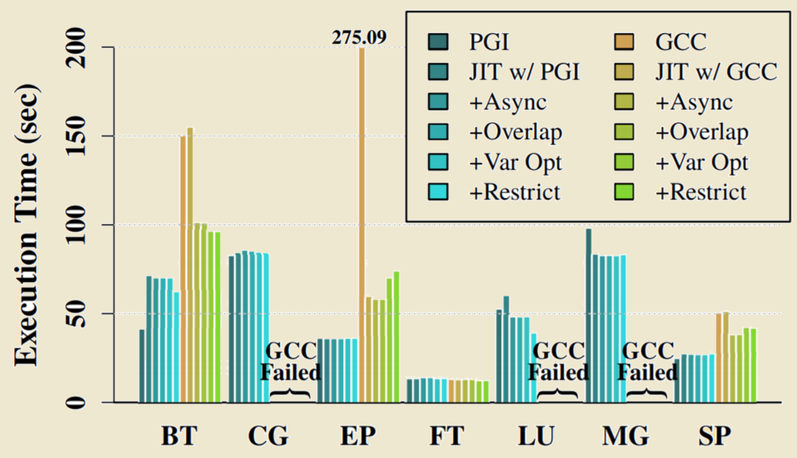
+Async: Single Queue
+Overlap: 16 Queues

Kernel Specialization

- Profiling Execution
- Additional Compilation

```
#define N 1024
void kernel0_opt(float (* restrict x),
                 float (* restrict y)) {
 /* ... */
}
```

+Var Opt: Replace Var with Const
+Restrict: Declare with Restrict



Manually-tuned NAS Parallel Benchmarks on NVIDIA DGX-1 w/ Tesla V100 SXM2 (16GB)

- **+Async** improves by 3.43% with PGI and 22.08% with GCC on average
- No improvement by **+Overlap** because of little inter-kernel parallelism
- **+Var Opt** worsens the performance due to less ILP
- **+Restrict** enables parallelized memory access with additional register use

03 MULTI-GPU UTILIZATION

- Previous work: loop splitting over plural GPUs or required manual efforts [3-4]
- Mere loop splitting causes *unsolvable dependencies* among devices in real applications
- Unified memory can be employed among GPUs, but memory thrashing is inevitable
- Key background to automate multi-GPU use
  1. Increased P2P-Bandwidth 2. Less overheads of computation compared to memory accesses

Predicate-Based Filtering

```
a[i]*=2;  →  (a_lb <= i && a_ub >= i) ? a[i]*=2 : a[i];
```

lb: lower-bound, ub: upper-bound

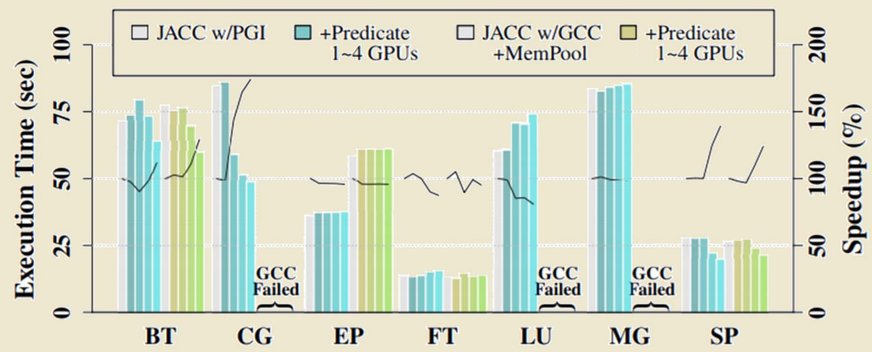
- We duplicate the program structure on all the GPUs
- Our technique limits memory accesses depending on data regions that the GPU writes to
- GPU-to-GPU communication follows after each kernel execution

JACC Runtime

Data Management	Kernel Execution
- Array Mapping	- Code Gen/Compile
- GPU-to-GPU or CPU-to/from-GPU Communication	- Parameter Calculation
	- Switching GPUs
	- Adaptive Running

Dynamic Code

```
void kernel0(…)
{ //Declarations
  //Predicated Kernel
  //Reduction
}
```



- We integrate adaptive execution for predicate-based filtering into JACC
- Our technique successfully parallelize BT/SP which previous work cannot
- The more GPUs are used, the more latencies are reduced for kernel/comm

04 CONCLUSION

- JACC is an OpenACC framework which facilitates runtime extension
- We enabled **asynchronous execution** / **on-the-fly kernel specialization** / **multi-GPU execution** without additional user effort
- Our technique to distribute practical applications improved the performance of some of manually-tuned NAS Parallel Benchmarks

[1] NVIDIA Corporation. 2017. NVIDIA Tesla V100 GPU Architecture. https://images.nvidia.com/content/volta-architecture/pdf/voltaarchitecture-whitepaper.pdf
[2] The OpenACC Organization. 2011. OpenACC. https://www.openacc.org/
[3] Kazuaki Matsumura et. al. 2018. MACC: An OpenACC transpiler for automatic multi-GPU use
[4] Jungwon Kim et. al. 2012. SnuCL: An OpenCL framework for heterogeneous CPU/GPU clusters

Kazuaki Matsumura | Simon Garcia de Gonzalo | Antonio J. Peña
Barcelona Supercomputing Center (BSC-CNS)
Email: {kazuaki.matsumura|simon.garcia|antonio.pena}@bsc.es

ISC 2021 (JUNE 24- JULY 2, 2021)

EPEEC PRODUCTIVITY AT EXASCALE
The EPEEC project has received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement N° 801051