PGI® COMPILERS &TOOLS

SC18 Dallas, TX | hpc inspires.

OPENMP GPU OFFLOAD IN FLANG AND LLVM

Guray Ozen, Simone Atzeni, Michael Wolfe

Annemarie Southwell, Gary Klimowicz

NVIDIA.

MOTIVATION

What does HPC programmer need today?

- Performance → **GPUs**, multi-cores, other accelerators.
- Continuity → **Fortran!** Current and future large codes.
- Quick Porting → **OpenMP**, OpenACC, MPI etc.
- Cross platform compiler → **LLVM**, PGI, GNU, etc.

Conclusion, Fortran programmers needs

OpenMP GPU Offload in Flang

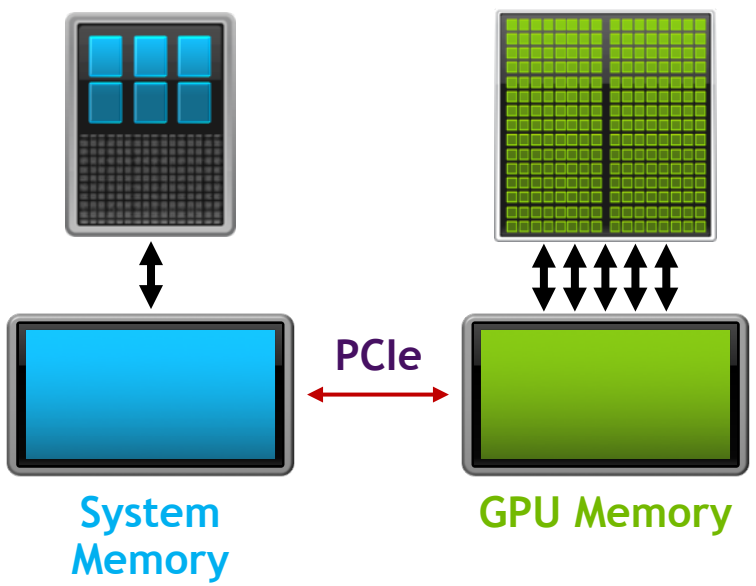
AGENDA

- Motivation
- **Background**
- Integration of OpenMP GPU Offload into Flang
- Experimental Evaluation
- Conclusion

Programming GPU-Accelerated Systems

Separate CPU System and GPU Memories

GPU Developer View



OPENMP 4.5

Device Offload Model

```
!$OMP TARGET ENTER DATA MAP(to:x, y)
```

copy array x, y
to the default device

```
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO
DO i=1, n
  y(i) = a * x(i) + y(i)
ENDDO
```

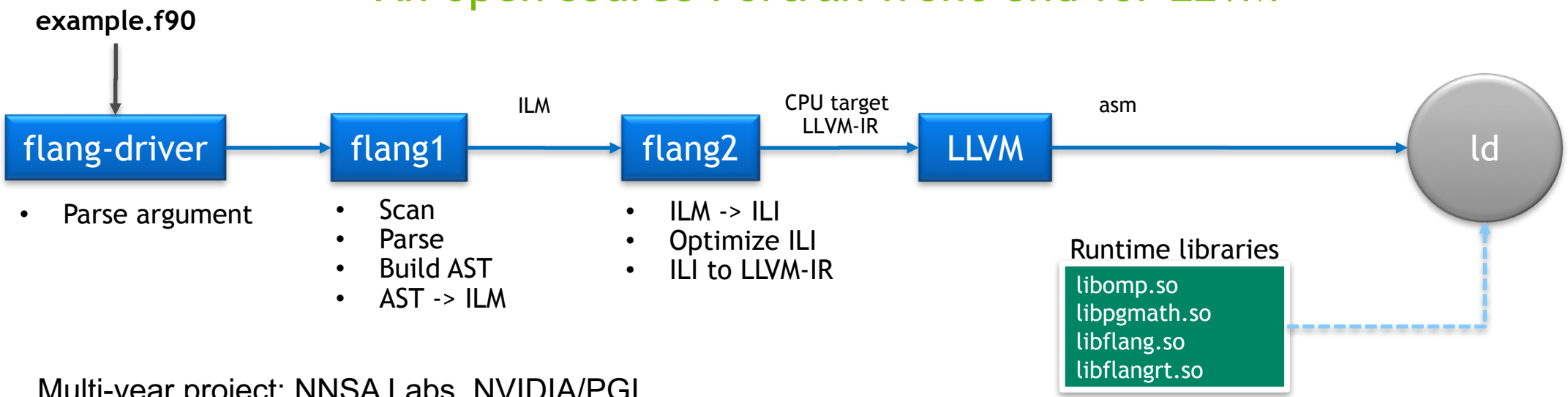
```
!$OMP TARGET EXIT DATA MAP(from: y)
```

copy array y
from the default
device

target:	Offload region
teams:	Create teams of threads
distribute:	Distribute the iteration across the master threads of all teams
parallel:	Create threads
do:	Distribute the iteration across the threads that already exist in the team

THE FLANG PROJECT

An open source Fortran front-end for LLVM



Multi-year project: NNSA Labs, NVIDIA/PGI

Based on the PGI Fortran front-end

Re-engineering for integration with LLVM

Develop CLANG-quality Fortran front end

Glossary
ILM: PGI-IR generated by frontend
ILI: PGI-IR generated by backend

PGI

OPENMP GPU OFFLOAD IN FLANG

Design Goals

Provide an implementation of OpenMP 4.5 GPU Offload targeting NVIDIA GPUs in Flang

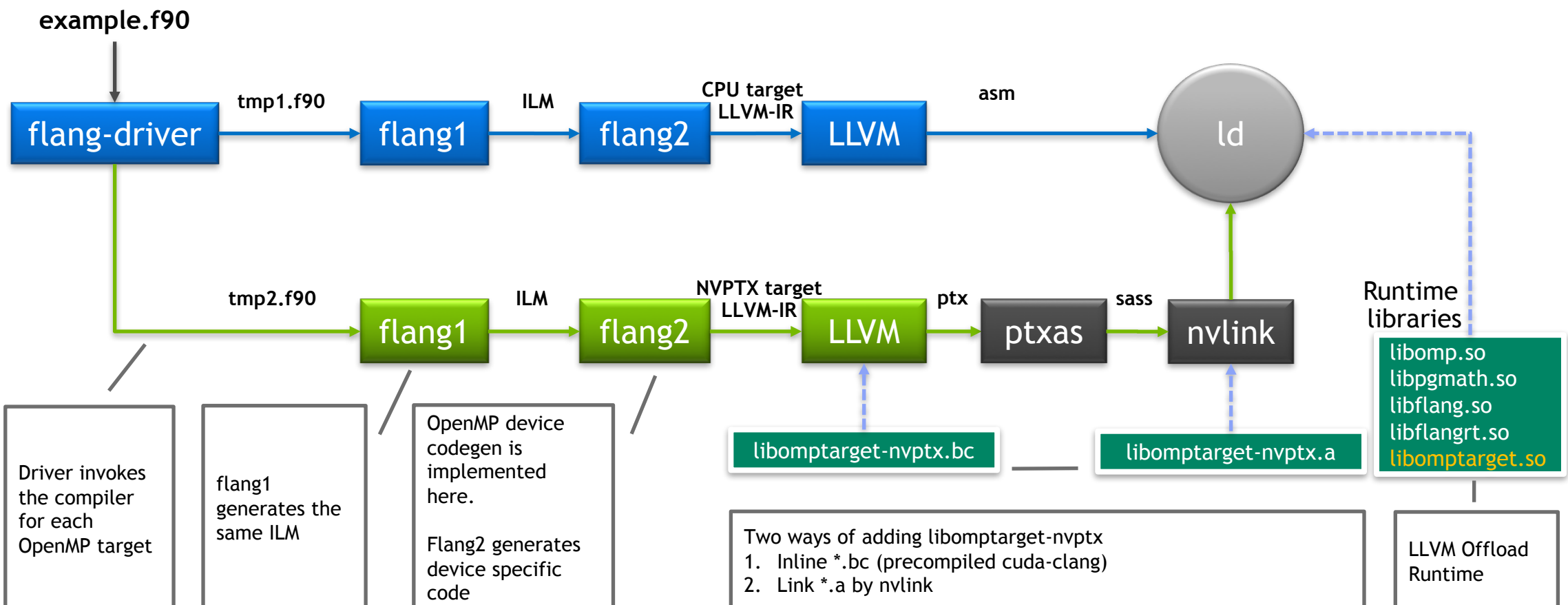
Aim to keep the same design as Clang OpenMP 4.5

- Leverage the Clang/LLVM driver
- Take advantage of LLVM's OpenMP runtimes

Compatibility with Flang and Clang OpenMP GPU Offload

COMPILATION PIPELINE

$ flang -fopenmp -fopenmp-targets=nvptx64-nvidia-cuda example.f90



CODE TRANSFORMATION

Example with OpenMP CPU

```
!$OMP TARGET TEAMS DISTRIBUTE
PARALLEL DO
DO i=1, n
  y(i) = a * x(i) + y(i)
ENDDO
```

- Use libomp(kmpc) as CPU OpenMP runtime
- Outline for each OpenMP construct

Generated LLVM-IR for Host target

```
define void @MAIN_ () {
  call void @targetFUNC(i32* %0, ...)
}
define internal void @targetFUNC(){
  call @__kmpc_fork_teams (... , %teamsFUNC)
  ret void
}
define internal void @teamsFUNC(){
  call void @__kmpc_for_static_init_4 (...)
  call void @__kmpc_fork_call(..., %parallelFUNC)
  call void @__kmpc_for_static_fini (...)
  ret void
}
define internal void @parallelFUNC(){
  call void @__kmpc_for_static_init_4 (...)
  ; <The loop body >
  call void @__kmpc_for_static_fini (...)
  ret void
}
```

Outlining for **$target**

Outlining for **$teams** & fork teams

Distribute loop across **$teams**

Outline for **$parallel** & fork threads

Distribute loop across **threads**

CODE TRANSFORMATION

Example with OpenMP GPU Offload

```
!$OMP TARGET TEAMS DISTRIBUTE
PARALLEL DO MAP(to:x) MAP(tofrom:y)
DO i=1, n
  y(i) = a * x(i) + y(i)
ENDDO
```

- Offload target region to the device
- New code transformation for device
- Use libomptarget for GPU offload
- Use libomptarget-nvptx in device OpenMP RT

Generated LLVM-IR for Host target

```
define void @MAIN_ () {
  call void @__tgt_target_teams(i8* %kernelID, ...)
}
```

Offload **$target** to GPU

Generated LLVM-IR for NVPTX target

```
define ptx_kernel void @kernel(){
  call @ teamsFUNC (...)
  ret void
}
define internal void @teamsFUNC(){
  call void @__kmpc_for_static_init_4 (...)
  call void @parallelFUNC(...)
  call void @__kmpc_for_static_fini (...)
  ret void
}
define internal void @parallelFUNC(){
  call void @__kmpc_for_static_init_4 (...)
  ; <The loop body >
  call void @__kmpc_for_static_fini (...)
  ret void
}
```

Outlining for **$teams** & call func

Distribute loop across **$teams**

Outline for **$parallel** & call func

Distribute loop across **threads**

PGI

OUTLINER OPTIMIZATION

Reduce number of outlining for device code

Calling function in the device is not efficient

Omit outlining for combined constructs in device!

Generated LLVM-IR - Default

```
define ptx_kernel void @kernel(){
  call @ teamsFUNC (...)
  ret void
}
define internal void @teamsFUNC(){
  call void @__kmpc_for_static_init_4 (...) !$distribute
  call void @parallelFUNC(...)
  call void @__kmpc_for_static_fini (...)
  ret void
}
define internal void @parallelFUNC(){
  call void @__kmpc_for_static_init_4 (...) !$do
  ; <The loop body >
  call void @__kmpc_for_static_fini (...)
  ret void
}
```

Generated LLVM-IR - Reduced number of outlining in device

```
define ptx_kernel void @kernel(){
   call void @__kmpc_for_static_init_4 (...) ; !$distribute

   call void @__kmpc_for_static_init_4 (...) ; !$do
    ; <The loop body >
   call void @__kmpc_for_static_fini (...)

   call void @__kmpc_for_static_fini (...)
}
```

PGI

EXPERIMENTAL EVALUATION

Compilers studied in our experiments

We run our experiments on IBM POWER8NVL and NVIDIA Pascal GPUs with CUDA 8.0

Compilers	Flags
Flang – OpenMP GPU (this work)	-O3 -fopenmp -fopenmp-targets=nvptx64-nvidia-cuda -Mpreprocess
Clang 7.0	-O3 -fopenmp -fopenmp-targets=nvptx64-nvidia-cuda
PGI 18.7	-Mpreprocess -fast -ta=tesla,cc60,cuda8.0 -mp
IBM XLF 16.0.1	-qsmp -qoffload -qpreprocessor -O3

EXPERIMENTAL EVALUATION (II)

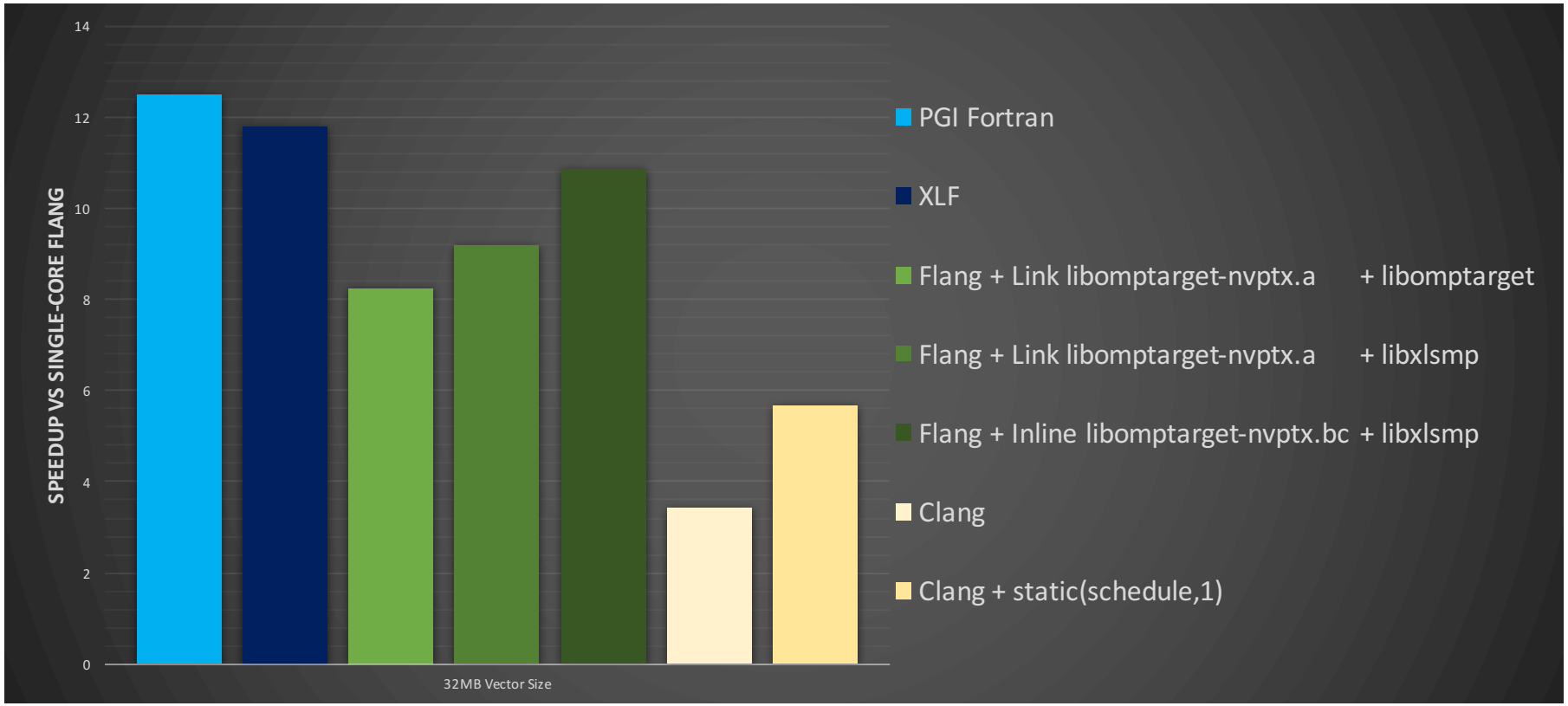
Compilers studied in our experiments

We experiment flang with different runtimes

Compiler	Target Offload Runtime	Device OpenMP Runtime
Flang	LLVM Offload RT - *libomptarget*	Compile device RT with nvcc Link *libomptarget-nvxptx*
Flang	IBM XL Offload RT - *libxlsmp*	Compile device RT with nvcc Link *libomptarget-nvxptx*
Flang	IBM XL Offload RT - *libxlsmp*	Compile device RT clang-cuda Inline *libomptarget-nvxptx.bc*

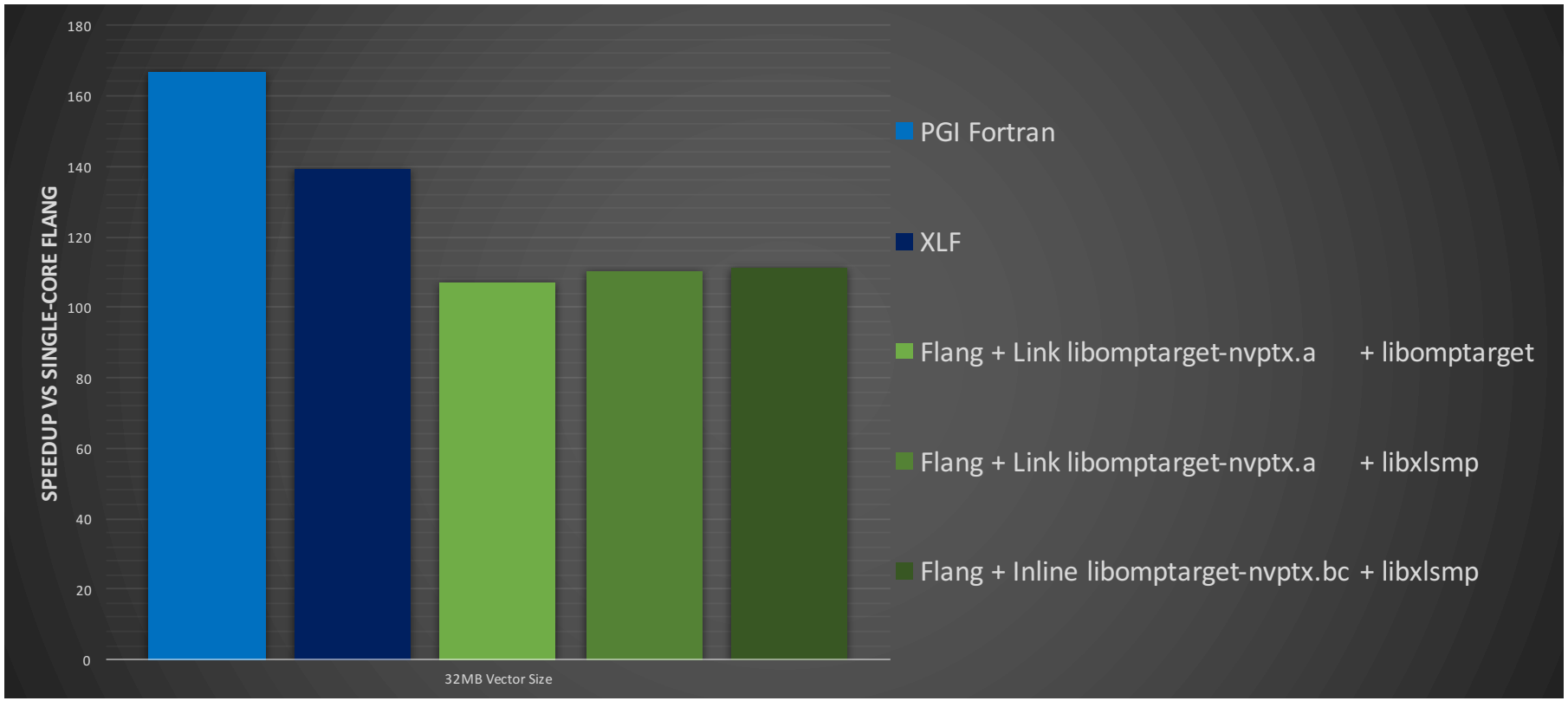
DAXPY

y = y *ax


SPEEDUP VS SINGLE-CORE FLANG
14
12
10
8
6
4
2
0
32MB Vector Size
PGI Fortran
XLF
Flang + Link libomptarget-nvptx.a + libomptarget
Flang + Link libomptarget-nvptx.a + libxlsmp
Flang + Inline libomptarget-nvptx.bc + libxlsmp
Clang
Clang + static(schedule,1)

360.ILBDC

Spec Accel



Performances are considered ESTIMATES per SPEC run and reporting rules.
SPEC® is a registered trademark of the Standard Performance Evaluation Corporation (www.spec.org).

CONCLUSION

Flang OpenMP GPU Offload

- Introduced OpenMP GPU Offload to Flang
  - Support for combined-constructs
- Proposed compatible implementation with Clang
- Flang performance is in line with Clang
  - Obtained better performance with IBM XL's target offload runtime


NVIDIA