TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU

The University of Texas at Austin

SCILIB-Accel: Performant Automatic BLAS Offload for NVIDIA Grace-Hopper

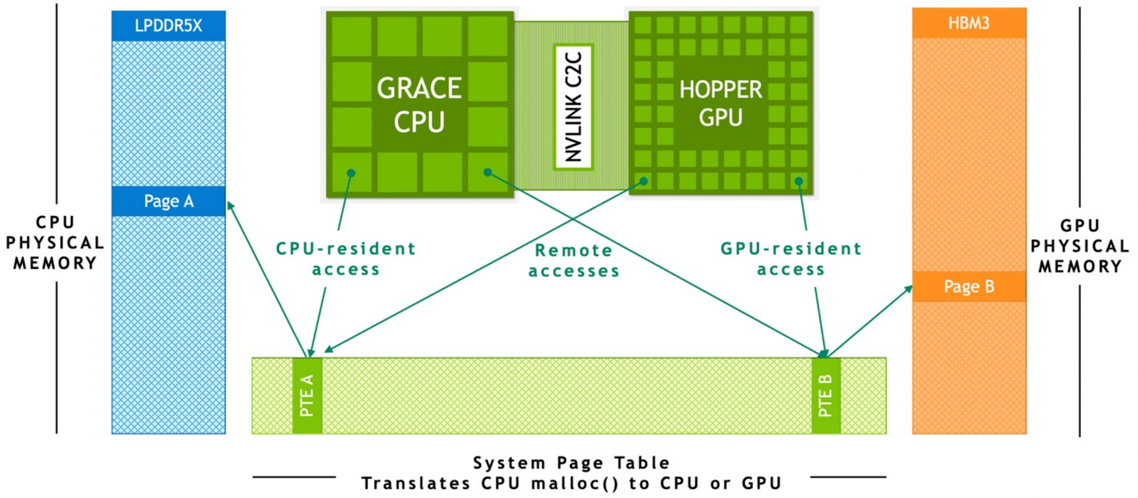
PRESENTED BY:

Junjie Li (jli@tacc.utexas.edu)

Texas Advanced Computing Center,

The University of Texas at Austin

Grace-Hopper: Key features



- single system page table for LPDDR5 + HBM
- conventional GPU managed page table for HBM
- coherent memory at cache line level
- fast C2C NVLink
- active research to utilize these new features

Allocator	Initial Page Table Entry	Cache Coherent
malloc	CPU	yes
cudaMallocManaged	CPU	yes
cudaMalloc	GPU	no
cudaMallocHost	CPU	no

GH200: Stream & HPL

STREAM TRIAD Bandwidth (GB/s)

	CPU	GPU
LPDDR5X	418	610
HBM	142	3680

Astonishing bandwidth.
Data locality still matters.

Highly FP64 capable GPU
Recent driver + recent cuBLAS: dgemm can use tensor core.

HPL (FP64)

	Rmax (Tflops)
Grace (72C)	2.8
Hopper	52.9

TACC

Porting PARSEC to GPU

- real-space DFT code at UT Austin by Prof. Chelikowsky
- PARSEC heavily relies on dgemm (77% runtime)
- Most dgemm comes from ScaLAPACK (pdgemm, pdtrsm)
- Never run on GPU before
- BLAS -> cuBLAS?
  - no way to rewrite ScaLAPACK
  - cuBLAS: different interface
  - no intent to rewrite user code
  - symbols interception and automatic offload ?

```
recorded MPI rank: 0
total runtime: 681.9s
--------------  function statistics (exclusive) -------------
    exclusive call time (in seconds) and counts
--------------------------------------------------------------
    |   group    |    function     |    count     |    time    |

  1 |       BLAS |          dgemm_ |        43481 |    524.512 |
  2 |      PBLAS |         pdtrsm_ |          750 |     25.975 |
  3 |      PBLAS |         pdgemm_ |         1505 |     13.446 |
  4 |      PBLAS |         pdsymv_ |        23840 |      4.680 |
  5 |       BLAS |          dcopy_ |     28194164 |      3.236 |
  6 |       BLAS |          dtrsm_ |          427 |      1.686 |
  7 |       BLAS |          dgemv_ |      1157095 |      1.389 |
 ...
 ...
```

Auto BLAS Offload Attempts

- Cray LIBSci (since Titan), IBM ESSL, NVidia NVBLAS (heavy overheads)
- Some require relink, or only work for dynamically linked BLAS
- **All performs mandatory data copies to/from GPU**
  - copy data costs more time than compute
- hardly useful in practice.

- **SCILIB-Accel: a new tool to overcome all these issues!**

SCILIB-Accel: workflow

SCILIB-Accel:
hacks user binary and
redirects BLAS call to a
custom-built BLAS wrapper

```
mydgemm( …) {
  move data to gpu
  call cublasDgemm(…)
  move data
  timer()
}
```

User application:

```
...
...
call dgemm(…)
…
```

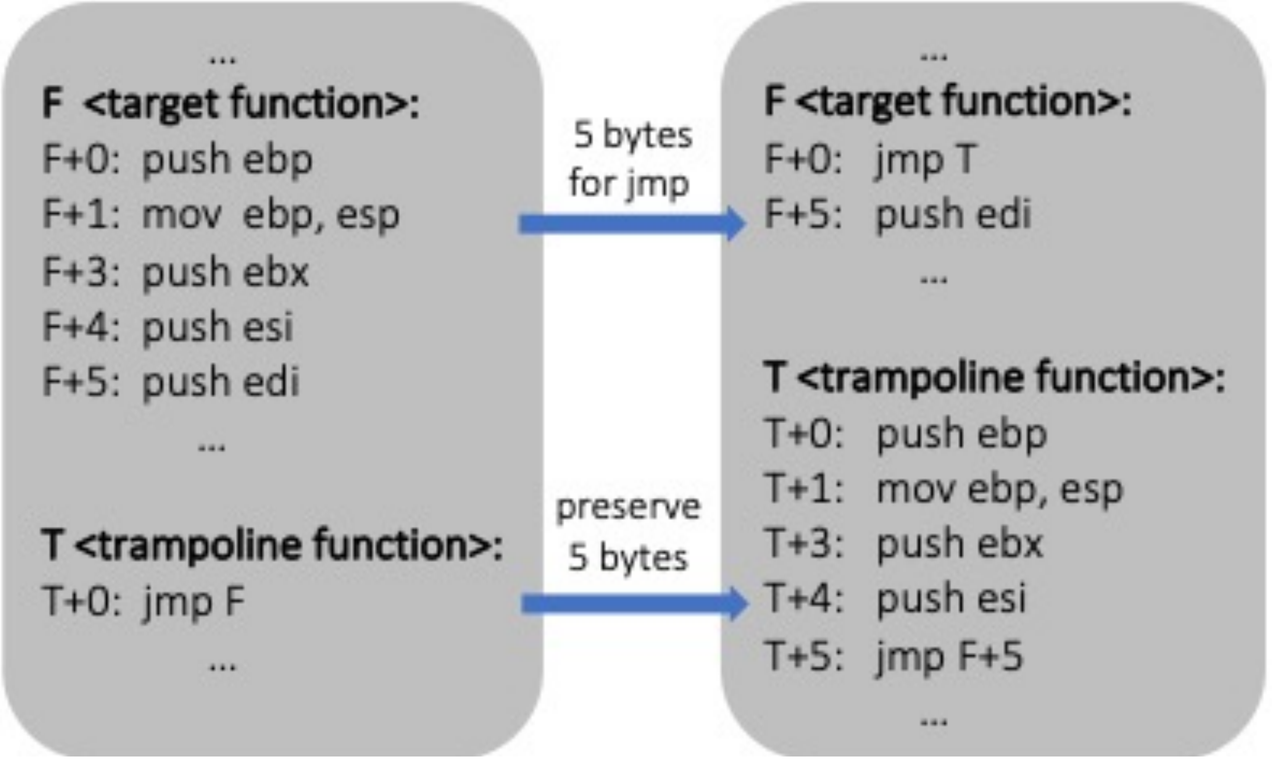
CPU BLAS Library

```
dgemm( …) {
  …
}
```

SCILIB-Accel: intercept BLAS calls

Replace BLAS calls with CUBLAS calls (level-3 for now)

- Symbol interception by **trampoline-based Dynamic Binary Instrumentation**
- No user code recompile
- Effective for both dynamically and statically linked BLAS.
- Negligible overhead when replacing function with same signature.



TACC

Yinzhi Wang and Junjie Li, "PEAK: a Light-Weight Profiler for HPC Systems", https://dl.acm.org/doi/abs/10.1145/3624062.3624143

SCILIB-Accel: data movements strategy 1/3

1. **Mem-copy**: cudaMemCpy Matrices between CPU mem to GPU mem for every CUBLAS call (all other tools do this)

```
cudaMemCpy matrix A, B, C to GPU

cublasDgemm(…, device_A, .., device_B, .., device_C, ..)

cudaMemCpy matrix C back to CPU
```

Method		application total runtime	dgemm time	data movement
CPU, single Grace		776.5s	608s	0
Auto offload	**S1: data copy**	**425.7s**	**12.4s**	**220.7s**

TACC

SCILIB-Accel vs NVBLAS

- NVBLAS is supposed to be equivalent to SCILIB-Accel Strategy 1
- but NVBLAS has ridiculous overhead, not usable at all.

Runtime of cublasDgemm w/ cudaMemCpy (transA='T', transB='N', M=32, N=2400, K=93536)

	GH200	H100-PCIe	GH200	H100-PCIe
Offload Strategy	Strategy 1	Strategy 1	NVBLAS	NVBLAS
Total time	5.50ms	32.80ms	54.8 ms	134.0ms
1. cudaMemcpy†	4.96 ms	31.79ms	-	-
2. cublasDgemm	0.52 ms	0.99ms	-	-
3. other	0.02 ms	0.02ms	-	-

† Including copying matrices A, B and C to GPU memory and C back to host memory.

TACC

J. Li, Y. Wang, X. Liang, H. Liu, "Automatic BLAS Offloading on Unified Memory Architecture: A Study on NVIDIA Grace-Hopper", https://arxiv.org/abs/2404.13195

SCILIB-Accel: data movements strategy 2/3

2. **Coherent Access:** just pass the CPU malloc pointer to GPU kernel
   let counter-based migration or compiler optimization (-gpu=unified) to move data

```
cublasDgemm(..., host_A, .., host_B, .., host_C, ..)
```

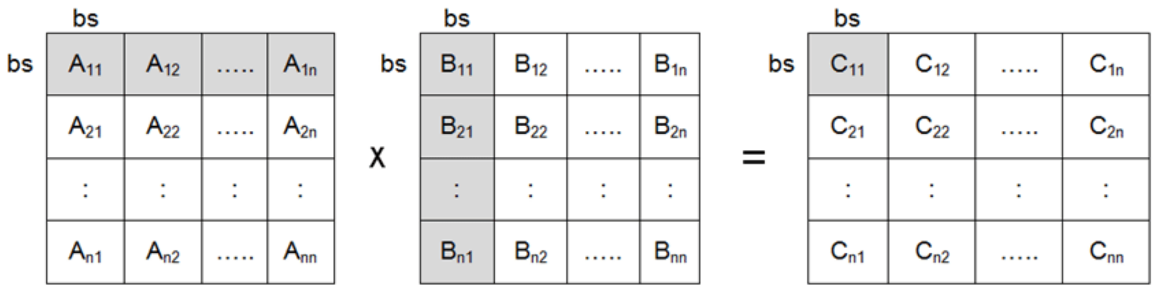
Method		application total runtime	dgemm time	data movement	
CPU, single Grace		776.5s	608s	0	
Auto offload	S1: data copy	425.7s	12.4s	220.7s	
	S2: cuda runtime	470.0s	234.0s	in dgemm	CUDA counter-based migration
	S2: -gpu=unified*	246.8s	56.6s	in dgemm	

* requires recompile of PARSEC and SCILIB-Accel

TACC

Strategy 3: Common level3 BLAS use patterns

- Case 1: block matrix multiplication, e.g. ScaLAPACK. Every submatrix of A multiplies with submatrix of B



- Case 2: Self-Consistent Field calculation, calculations are performed on the same matrix until convergence is reached.

- Case 3: a sequence of matrix multiplications. Each matrix is involved in multiply gemm calls

A x B = C
other codes accessing C..
C x D = E
..

- In these common cases, matrix inputs for gemm calls can be re-used
- LDDDR5 in NUMA0, HBM in NUMA1
- There will be CPU accessing the matrices (normalize, scale, trace, etc….)
  - through coherent access on GH
- **Assume non-gemm CPU access of the matrices are relatively trivial, matrices can be kept resident on NUMA 1 (just like remote memory access is assumed to be relatively trivial in OpenMP first touch)**
- leads to Strategy 3:
  GPU first use

SCILIB-Accel: data movements strategy 3/3

3. **GPU First Use**: matrices are moved HBM the first time they are used by CUBLAS.
   - Matrices stay on HBM throughout their lifetime
   - Implemented by migrating memory pages from NUMA0 to NUMA1, move_page function from libnuma is used.

```
if (on numa 0) move_page (host_A) ;

if (on numa 0) move_page (host_B) ;

if (on numa 0) move_page (host_C) ;

cublasDgemm(..., host_A, .., host_B, .., host_C, ..)
```

TACC

J. Li, Y. Wang, X. Liang, H. Liu, "Automatic BLAS Offloading on Unified Memory Architecture: A Study on NVIDIA Grace-Hopper", https://arxiv.org/abs/2404.13195

SCILIB-Accel: easy to use

Just LD_PRELOAD it!
All level-3 BLAS are implemented!

```
LD_PRELOAD=$PATH_TO_LIB/scilib-dbi.so
run your cpu code.
```

get code from here

https://github.com/nicejunjie/scilib-accel

TACC

SCILIB-Accel: data movements strategy 3

First Use Policy, best performance!
4k page: 220.3 , 64k page: 230s

PARSEC performantly runs on GPU for the first time ever!

Matrix reuse: every matrix moved to HBM gets reused 570 times on average by subsequent dgemm calls.

Method		application total runtime	dgemm time	data movement
CPU, single Grace		776.5s	608s	0
Auto offload	S1: data copy	425.7s	12.4s	220.7s
	S2: CUDA runtime	470.0s	234.0s	in dgemm
	S2: -gpu=unified*	246.8s	56.6s	in dgemm
	S3: GPU First Use	**220.3s**	**29.1s**	**1.3s**

Matrix reuse: 570

* requires recompile of PARSEC and SCILIB-Accel

Application Tests: MuST

MuST suite solves the Kohn-Sham equation by solving the Green's function.
In contrast to solving the wave-function, it can perform KKR-CPA calculations for random structures and LSMS calculations for large systems with linear scalability to the system size.
https://github.com/mstsuite/MuST

The code has a native GPU version calling cuSolver doing the critical matrix inverse.
SCILIB-Accel auto offload does much better offloading more components to GPU.

MuST

Method		App Runtime	zgemm+ztrsm time	data movement time	
CPU, Grace		124s	82.5 + 35.2s	-	
Auto offload	GPU First Use	30.7s	15.9s+3.8s	3.6s	**Matrix reuse: 70**

SCILIB-Accel: Performant Auto BLAS offload

- High bandwidth C2C NVLink and coherent memory inspires new possibilities of doing auto offload.
- https://github.com/nicejunjie/scilib-accel
- automatically offload CPU BLAS calls to GPU
- no user code modification or recompilation
- Works for statically and dynamically linked BLAS
- Better performance than all other tools

Final comment

- Easy way to explore GPU capability
- Get incentives to port CPU codes to GPU
- Performance may further improve as there is still performance issue of CUDA kernel accessing malloc HBM memory.
- Let me know if this tool helps you.

https://github.com/nicejunjie/scilib-accel

TACC