PETSc on GPUs and MIC: Current Status and Future Directions

Karl Rupp

Institute for Microelectronics
Institute for Analysis and Scientific Computing
TU Wien, Austria

Celebrating 20 Years of Computational Science with PETSc
Why bother?

GFLOPs/Watt

Peak Floating Point Operations per Watt, Double Precision

- CPUs, Intel
- GPUs, NVIDIA
- GPUs, AMD
- MIC, Intel

End of Year

GFLOP/sec per Watt

Why bother?

Procurements

Theta (ANL, 2016): 2nd generation INTEL Xeon Phi
Summit (ORNL, 2017), Sierra (LLNL, 2017): NVIDIA Volta GPU
Aurora (ANL, 2018): 3rd generation INTEL Xeon Phi
PETSc on GPUs and MIC:

Current Status
Available Options

Native on Xeon Phi
  Cross-compile for Xeon Phi

CUDA
  CUDA-support through CUSP
  \(-\text{vec\_type} \text{ cusp} \ -\text{mat\_type} \text{ aijcusp}\)
  Only for NVIDIA GPUs

OpenCL
  OpenCL-support through ViennaCL
  \(-\text{vec\_type} \text{ viennacl} \ -\text{mat\_type} \text{ aijviennacl}\)
  OpenCL on Xeon Phi very poor
CUDA (CUSP)

CUDA-enabled configuration (minimum)

./configure [..] --with-cuda=1
--with-cusp=1 --with-cusp-dir=/path/to/cusp

Customization:

--with-cudac=/path/to/cuda/bin/nvcc
--with-cuda-arch=sm_20

OpenCL (ViennaCL)

OpenCL-enabled configuration

./configure [..] --download-viennacl
--with-opencl-include=/path/to/OpenCL/include
--with-opencl-lib=/path/to/libOpenCL.so
How Does It Work?

Host and Device Data

```c
struct _p_Vec {
    ...
    void *data;       // host buffer
    PetscCUSPFlag valid_GPU_array;   // flag
    void *spptr;      // device buffer
};
```

Possible Flag States

```c
typedef enum {PETSC_CUSP_UNALLOCATED,
              PETSC_CUSP_GPU,
              PETSC_CUSP_CPU,
              PETSC_CUSP_BOTH} PetscCUSPFlag;
```
How Does It Work?

Fallback-Operations on Host

Data becomes valid on host (**PETSC_CUSP_CPU**)

```c
PetscErrorCode VecSetRandom_SeqCUSP_Private(..) {
    VecGetArray(...);
    // some operation on host memory
    VecRestoreArray(...);
}
```

Accelerated Operations on Device

Data becomes valid on device (**PETSC_CUSP_GPU**)

```c
PetscErrorCode VecAYPX_SeqCUSP(..) {
    VecCUSPGetArrayReadWrite(...);
    // some operation on raw handles on device
    VecCUSPRestoreArrayReadWrite(...);
}
```
Example

KSP ex12 on Host

$> ./ex12
   -pc_type ilu -m 200 -n 200 -log_summary

KSPGMRESOrthog  228 1.0 6.2901e-01
KSPSolve  1 1.0 2.7332e+00

KSP ex12 on Device

$> ./ex12 -vec_type cusp -mat_type aijcusp
   -pc_type ilu -m 200 -n 200 -log_summary

[0]PETSC ERROR: MatSolverPackage petsc does **not** support matrix type seqaijcusp
Example

KSP ex12 on Host

```bash
$> ./ex12
   -pc_type none -m 200 -n 200 -log_summary

KSPGMRESOrthog  1630  1.0  4.5866e+00
KSPSolve        1  1.0  1.6361e+01
```

KSP ex12 on Device

```bash
$> ./ex12 -vec_type cusp -mat_type aijcusp
   -pc_type none -m 200 -n 200 -log_summary

MatCUSPCopyTo  1  1.0  5.6108e-02
KSPGMRESOrthog  1630  1.0  5.5989e-01
KSPSolve        1  1.0  1.0202e+00
```
Pitfalls

**Pitfall: Repeated Host-Device Copies**

PCI-Express transfers kill performance
Complete algorithm needs to run on device
Problematic for explicit time-stepping, etc.

**Pitfall: Wrong Data Sizes**

Data too small: Kernel launch latencies dominate
Data too big: Out of memory

**Pitfall: Function Pointers**

Impossible to provide function pointers through library boundaries
OpenCL: Pass kernel sources, user-data hard to pass
Composability?
## Current GPU-Functionality in PETSc

<table>
<thead>
<tr>
<th></th>
<th>CUSP</th>
<th>ViennaCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Model</td>
<td>CUDA</td>
<td>OpenCL</td>
</tr>
<tr>
<td>Operations</td>
<td>Vector, MatMult</td>
<td>Vector, MatMult</td>
</tr>
<tr>
<td>Matrix Formats</td>
<td>CSR, ELL, HYB</td>
<td>CSR</td>
</tr>
<tr>
<td>Preconditioners</td>
<td>SA-AMG, BiCGStab</td>
<td>-</td>
</tr>
<tr>
<td>MPI-related</td>
<td>Scatter</td>
<td>-</td>
</tr>
</tbody>
</table>

### Additional Functionality

- MatMult via cuSPARSE
- OpenCL residual evaluation for PetscFE
PETSc on GPUs and MIC:
Future Directions
Future: CUDA

Split CUDA-buffers from CUSP

- Vector operations by cuBLAS
- MatMult by different packages
- CUSP (and others) provides add-on functionality

CUSPbuffers

CUSP ViennaCL

More CUSP Functionality in PETSc

- Relaxations (Gauss-Seidel, SOR)
- Polynomial preconditioners
- Approximate inverses
Future: PETSc + ViennaCL

ViennaCL

CUDA, OpenCL, OpenMP backends
Backend switch at runtime
Only OpenCL exposed in PETSc
Focus on shared memory machines

Recent Advances

Pipelined Krylov solvers
Fast sparse matrix-vector products
Fast sparse matrix-matrix products
Fine-grained algebraic multigrid
Fine-grained parallel ILU
Future: PETSc + ViennaCL

Current Use of ViennaCL in PETSc

$> ./ex12 -vec_type viennacl -mat_type aijviennacl ...

Executes on OpenCL device

Future Use of ViennaCL in PETSc

$> ./ex12 -vec_type viennacl -mat_type aijviennacl -viennacl_backend openmp,cuda ...

Pros and Cons

Use CPU + GPU simultaneously
Non-intrusive, use plugin-mechanism
Non-optimal in strong-scaling limit
Gather experiences for best long-term solution
Upcoming PETSc+ViennaCL Features

Pipelined CG Method, Exec. Time per Iteration

AMD FirePro W9100

NVIDIA Tesla K20m

ViennaCL 1.6.2 PARALUTION 0.7.0 MAGMA 1.5.0 CUSP 0.4.0
Upcoming PETSc+ViennaCL Features

Sparse Matrix-Vector Multiplication

ViennaCL 1.7.0
CUSPARSE 7
CUSP 0.5.1

Tesla K20m

GFLOP/sec

0 5 10 15 20

cantilever economics epidemiology harbor protein qcd ship spheres windtunnel accelerator
amazon0312 ca−CondMat cl−Patents circuit email−Enron p2p−Gnutella31 roadNet−CA webbase1m web−Google wiki−Vote
Upcoming PETSc+ViennaCL Feature

Sparse Matrix-Matrix Products

- ViennaCL 1.7.0, FirePro W9100
- ViennaCL 1.7.0, Tesla K20m
- CUSPARSE 7, Tesla K20m
- CUSP 0.5.1, Tesla K20m
- ViennaCL 1.7.0, Xeon E5−2670v3
- MKL 11.2.1, Xeon E5−2670v3
- ViennaCL 1.7.0, Xeon Phi 7120
- MKL 11.2.1, Xeon Phi 7120
Algebraic Multigrid Preconditioners

Total Solver Execution Times, Poisson Equation in 2D

- Dual INTEL Xeon E5-2670 v3, No Preconditioner
- Dual INTEL Xeon E5-2670 v3, Smoothed Aggregation
- AMD FirePro W9100, No Preconditioner
- AMD FirePro W9100, Smoothed Aggregation
- NVIDIA Tesla K20m, No Preconditioner
- NVIDIA Tesla K20m, Smoothed Aggregation
- INTEL Xeon Phi 7120, No Preconditioner
- INTEL Xeon Phi 7120, Smoothed Aggregation
Pipelined Solvers

Fine-Grained Parallel ILU (Chow and Patel, SISC, 2015)

Total Solver Execution Times, Poisson Equation in 2D

- Dual INTEL Xeon E5-2670 v3, No Preconditioner
- Dual INTEL Xeon E5-2670 v3, Fine-Grained ILU
- AMD FirePro W9100, No Preconditioner
- AMD FirePro W9100, Fine-Grained ILU
- NVIDIA Tesla K20m, No Preconditioner
- NVIDIA Tesla K20m, Fine-Grained ILU
- INTEL Xeon Phi 7120, No Preconditioner
- INTEL Xeon Phi 7120, Fine-Grained ILU

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)

Unknowns

Execution Time (sec)
Summary and Conclusion

Currently Available

CUSP for CUDA, ViennaCL for OpenCL
Automatic use for vector operations and SpMV
Smoothed Agg. AMG via CUSP

Next Steps

Use of cuBLAS and cuSPARSE
Better support for $n > 1$ processes
ViennaCL as CUDA/OpenCL/OpenMP-hydra