# PETSc on GPUs and MIC:
# Current Status and Future Directions

Karl Rupp

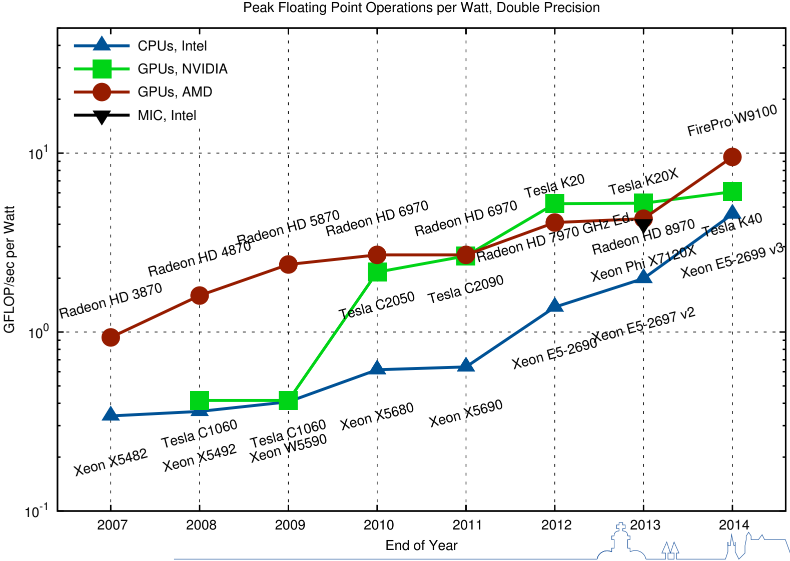
Institute for Microelectronics
Institute for Analysis and Scientific Computing

TU Wien, Austria

Celebrating 20 Years of Computational Science with PETSc

# Why bother?

## GFLOPs/Watt



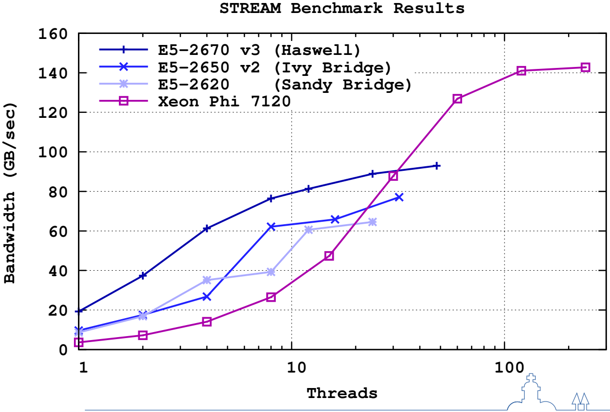
# Why bother?

## 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

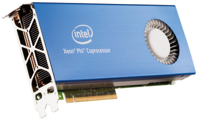


PETSc on GPUs and MIC:

Current Status

# Available Options

## Native on Xeon Phi




Cross-compile for Xeon Phi

## CUDA

CUDA-support through CUSP

`-vec_type cusp -mat_type aijcusp`

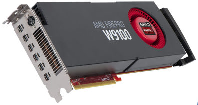
Only for NVIDIA GPUs


## OpenCL

OpenCL-support through ViennaCL

`-vec_type viennacl -mat_type aijviennacl`

OpenCL on Xeon Phi very poor



# Configuration

## 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

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

## Possible Flag States

```
  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`)

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

## Accelerated Operations on Device

Data becomes valid on device (`PETSC_CUSP_GPU`)

```
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

```
$> ./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

```
$> ./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

## Current GPU-Functionality in PETSc

| | **CUSP** | **ViennaCL** |
|---|---|---|
| Programming Model | CUDA | OpenCL |
| Operations | Vector, MatMult | Vector, MatMult |
| Matrix Formats | CSR, ELL, HYB | CSR |
| Preconditioners | SA-AMG, BiCGStab | - |
| MPI-related | Scatter | - |

## Additional Functionality

MatMult via cuSPARSE

OpenCL residual evaluation for PetscFE

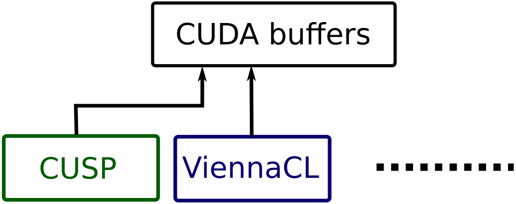
PETSc on GPUs and MIC:

Future Directions

# Future: CUDA

## Split CUDA-buffers from CUSP

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



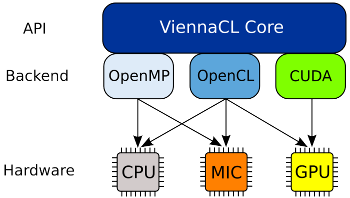

## More CUSP Functionality in PETSc

- 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

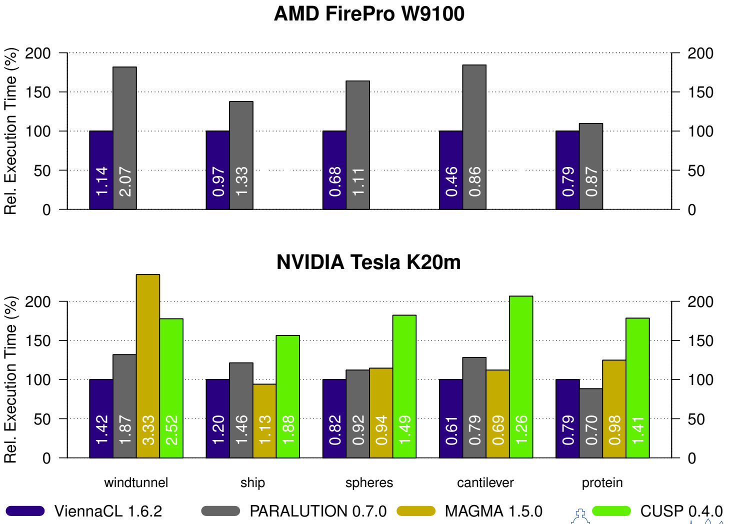
Non-intrusive, use plugin-mechanism

Non-optimal in strong-scaling limit

Gather experiences for best long-term solution

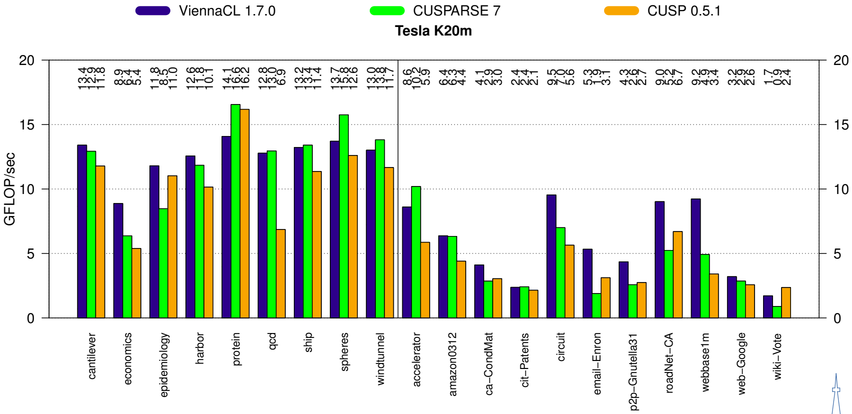
# Upcoming PETSc+ViennaCL Features

## Pipelined CG Method, Exec. Time per Iteration



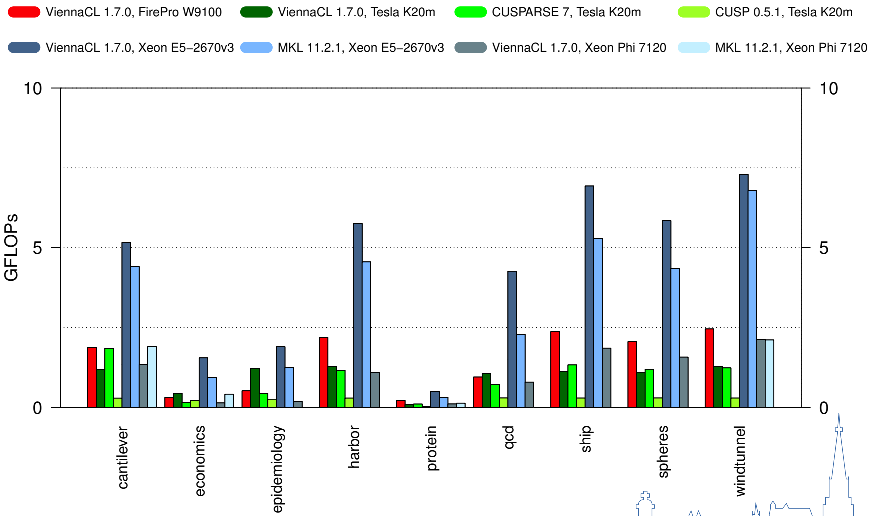
# Upcoming PETSc+ViennaCL Features

## Sparse Matrix-Vector Multiplication



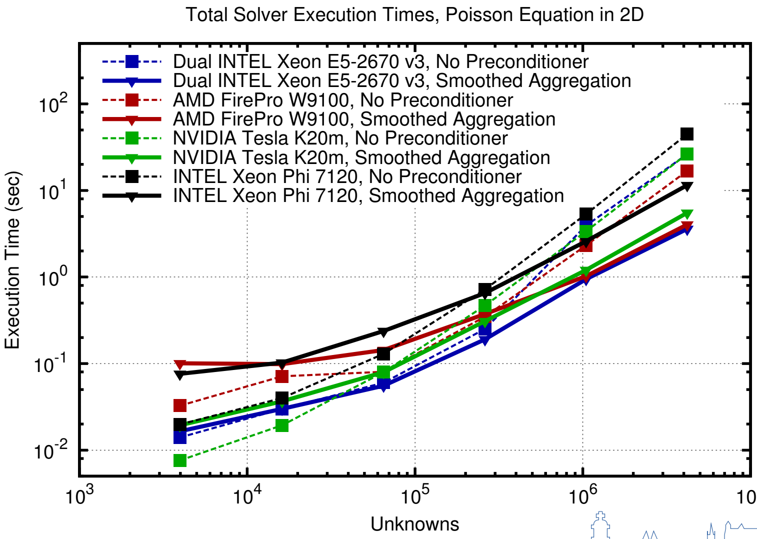
# Upcoming PETSc+ViennaCL Feature

## Sparse Matrix-Matrix Products



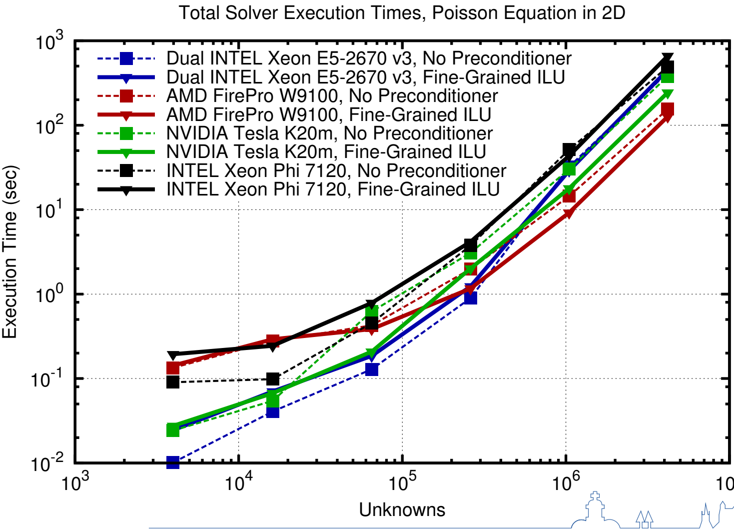
# Upcoming PETSc+ViennaCL Feature

## Algebraic Multigrid Preconditioners



# Pipelined Solvers

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



# 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