PETSc Tutorial

GPU and Manycore CPU Support in PETSc 3.9

GPU Part

Karl Rupp (complementing Richard Mills *et al.*)

rupp@iue.tuwien.ac.at

https://www.karlrupp.net/

Institute for Microelectronics, TU Wien

Imperial College London, UK

June 4-6, 2018

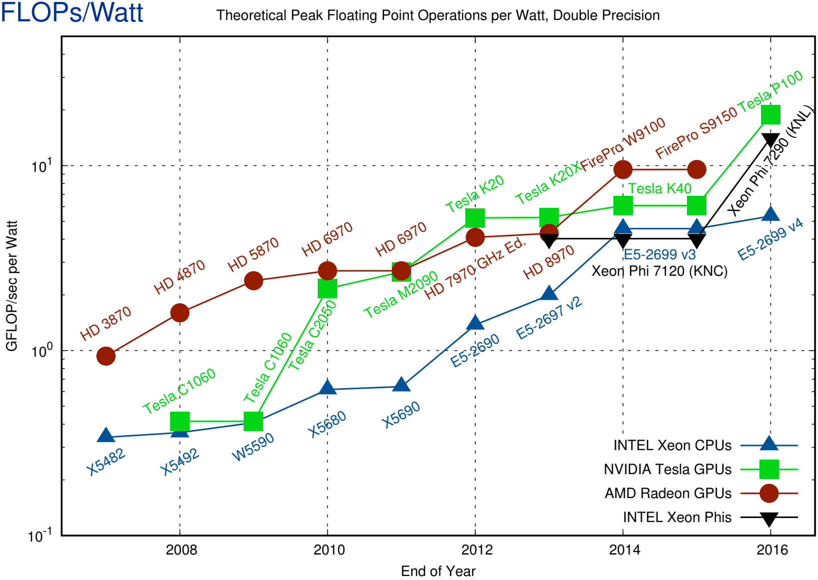
Why bother?

Don't believe anything
unless you can run it

Matt Knepley

Why bother?

GFLOPs/Watt



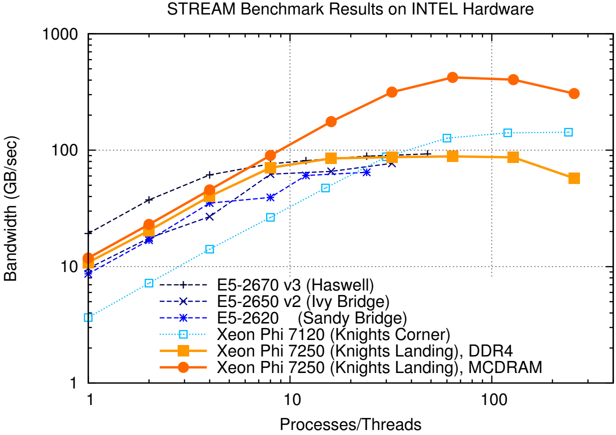
Why bother?

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~~

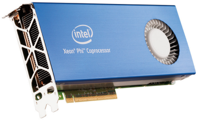


PETSc on GPUs and MIC:

Current Status

Available Options

Native on Xeon Phi



Forget KNC (“old Xeon Phi”)

“Just works” on KNL

CUDA

CUDA-support through CUDA/CUSPARSE

`-vec_type cuda -mat_type aijcusparse`

Only for NVIDIA GPUs

CUDA/OpenCL/OpenMP

CUDA/OpenCL/OpenMP-support through ViennaCL

`-vec_type viennacl -mat_type aijviennacl`

OpenCL on CPUs and MIC fairly poor

Configuration

CUDA (CUSPARSE)

```
./configure [..] --with-cuda=1
```

Customization:

```
  --with-cudac=/path/to/cuda/bin/nvcc
  --with-cuda-arch=sm_60
```

ViennaCL

```
./configure [..] --download-viennacl
```

Optional: CUDA/OpenCL/OpenMP

```
   --with-cuda=1
```

```
   --with-opencl-include=/path/to/OpenCL/include
   --with-opencl-lib=/path/to/libOpenCL.so
```

```
   --with-openmp=1
```

How Does It Work?

Host and Device Data

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

Possible Flag States

```
  typedef enum {PETSC_CUDA_UNALLOCATED,
                PETSC_CUDA_GPU,
                PETSC_CUDA_CPU,
                PETSC_CUDA_BOTH} PetscCUDAFlag;
```

How Does It Work?

Fallback-Operations on Host

Data becomes valid on host (`PETSC_CUDA_CPU`)

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

Accelerated Operations on Device

Data becomes valid on device (`PETSC_CUDA_GPU`)

```
PetscErrorCode VecAYPX_SeqCUDA(..) {
  VecCUDAGetArrayReadWrite(...);
  // some operation on raw handles on device
  VecCUDARestoreArrayReadWrite(...);
}
```

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 viennacl -mat_type aijviennacl
    -pc_type ilu -m 200 -n 200 -log_summary
```

```
[0]PETSC ERROR: MatSolverPackage petsc does not support
    matrix type seqaijviennacl
```

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 viennacl -mat_type aijviennacl
    -pc_type none -m 200 -n 200 -log_summary
```

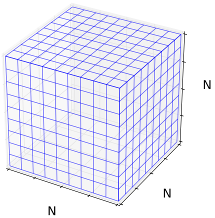
```
MatCUSPCopyTo          1 1.0 5.6108e-02
KSPGMRESOrthog      1630 1.0 5.5989e-01
KSPSolve               1 1.0 1.0202e+00
```

Pitfalls

Pitfall 1: GPUs are too fast for PCI-Express

Latest GPU peaks: 720 GB/sec from GPU-RAM, 16 GB/sec for PCI-Express

40x imbalance (!)



Compute vs. Communication

Take $N = 512$, so each field consumes 1 GB of GPU RAM

Boundary communication: $2 \times 6 \times N^2$: 31 MB

Time to load field: 1.4 ms

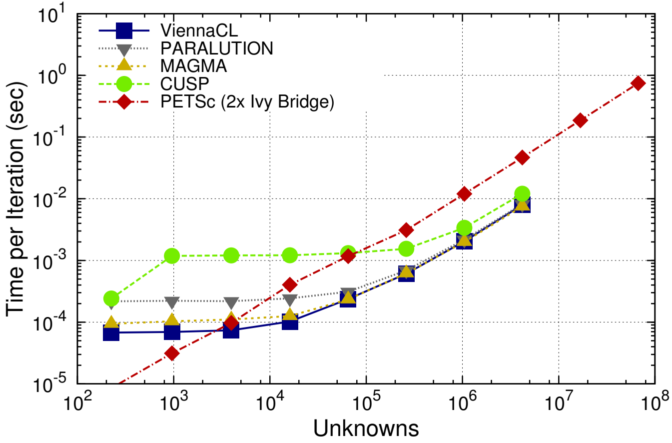
Time to load ghost data: **1.9 ms (!!)**

Pitfall 2: Wrong Data Sizes

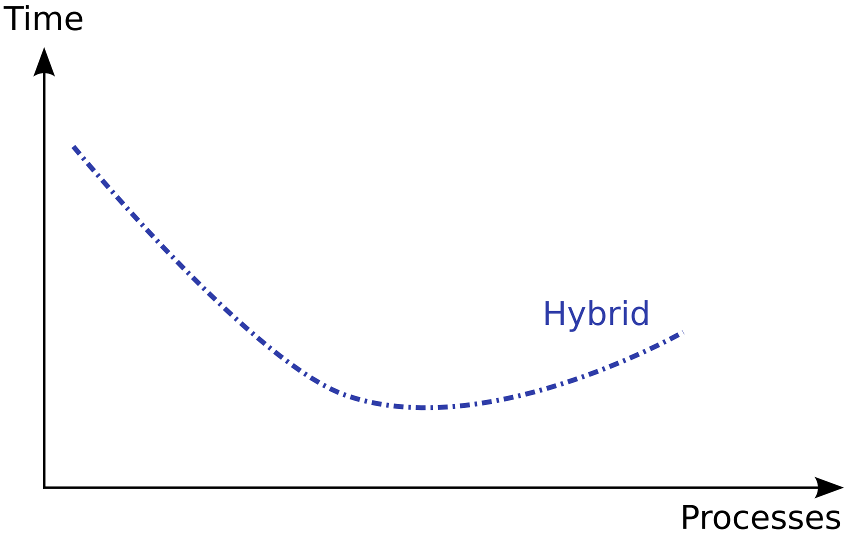
Data too small: Kernel launch latencies dominate

Data too big: Out of memory

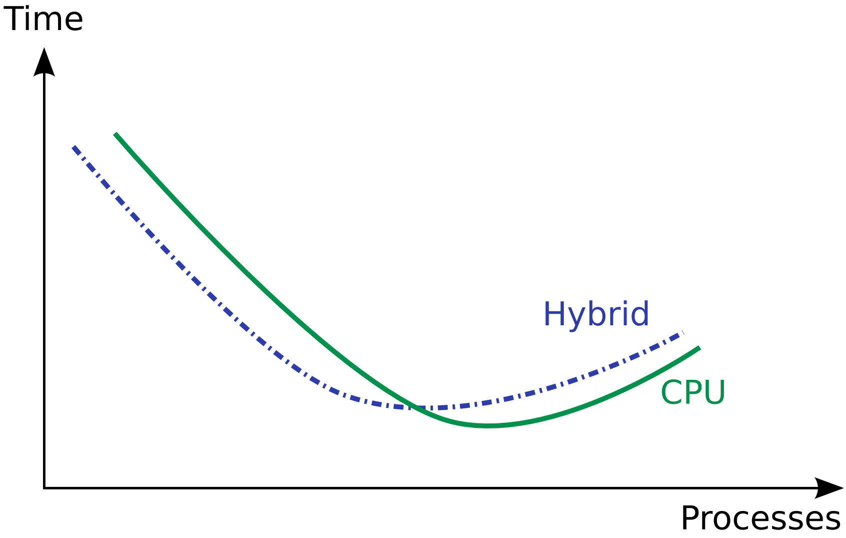
Unpreconditioned Conjugate Gradients, NVIDIA Tesla K20m



Strong Scaling Implications



Strong Scaling Implications



Pitfall 3: Composability of GPU codes

How to pass GPU pointers through library boundaries efficiently?

High-level interfaces tend to be polluted by low-level details

Many Non-Trivial PETSc Operations

do NOT benefit from modern high-end GPUs

in a substantial way!

(OpenPower systems can be exceptions)

Current GPU-Functionality in PETSc

	CUDA/CUSPARSE	**ViennaCL**
Programming Model	CUDA	CUDA/OpenCL/OpenMP
Operations	Vector, MatMult	Vector, MatMult
Matrix Formats	CSR, ELL, HYB	CSR
Preconditioners	ILU0	SA/Agg-AMG, Par-ILU0
MPI-related	Scatter	-

Additional Functionality

MatMult via cuSPARSE

OpenCL residual evaluation for PetscFE

GPU support for SuperLU-dist

GPU support for SuiteSparse

Current: PETSc + ViennaCL

Previous Use of ViennaCL in PETSc

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

Executes on OpenCL device

New Use of ViennaCL in PETSc

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

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

PETSc on GPUs and MIC:

Upcoming Features for the Next PETSc Release

Current Directions

GPU-acceleration for GAMG

- Use GPUs on the finest levels only
- Coarse grid solvers unchanged

ViennaCL: Better support for $n > 1$ processes

- Efficient scatter operations
- Minimize PCI-Express traffic

Bindings for NVIDIA's AMGX package

- AMGX is now available as open source
- GPU-alternative to GAMG, Hypre, ML

Conclusions

PETSc can help You

- solve algebraic and DAE problems in your application area
- rapidly develop efficient parallel code, can start from examples
- develop new solution methods and data structures
- debug and analyze performance
- advice on software design, solution algorithms, and performance

`petsc-{users,dev,maint}@mcs.anl.gov`

You can help PETSc

- report bugs and inconsistencies, or if you think there is a better way
- tell us if the documentation is inconsistent or unclear
- consider developing new algebraic methods as plugins, contribute if your idea works