PETSc Tutorial

PETSc and Many-Core/GPU Architectures

Karl Rupp

me@karlrupp.net

Freelance Computational Scientist
and
Institute for Microelectronics, TU Wien

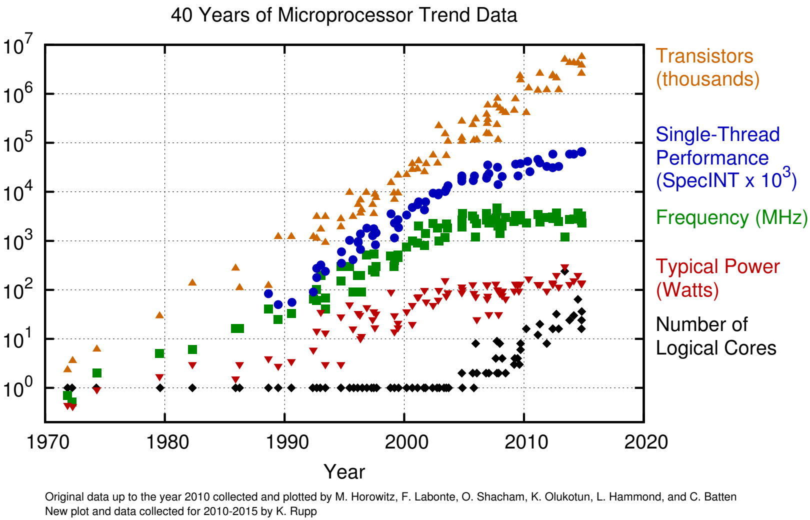
Boulder, Colorado

June 14-16, 2017

Table of Contents

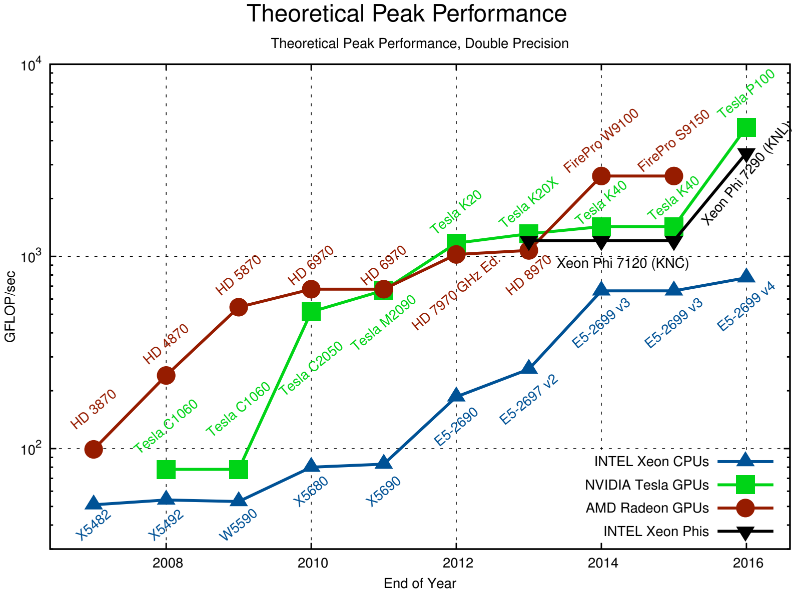
FLOPs and Bandwidth

Introduction



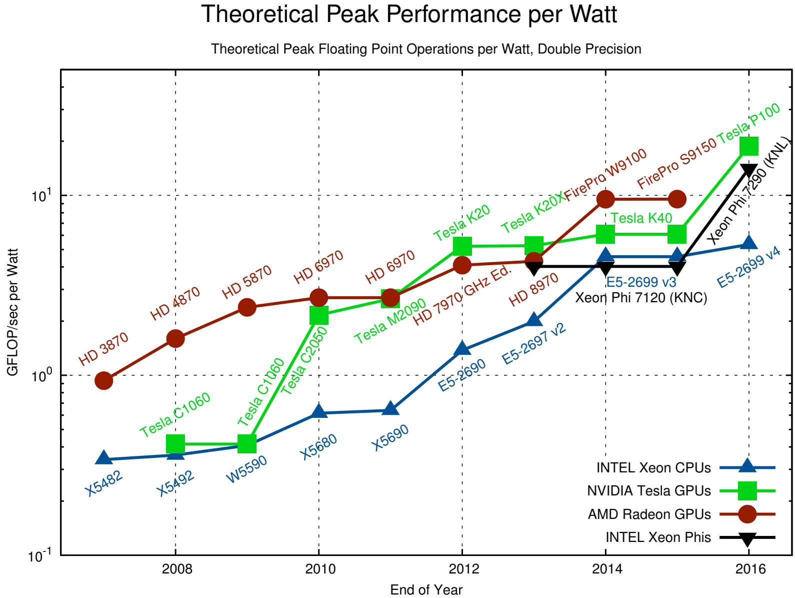
https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/

Introduction



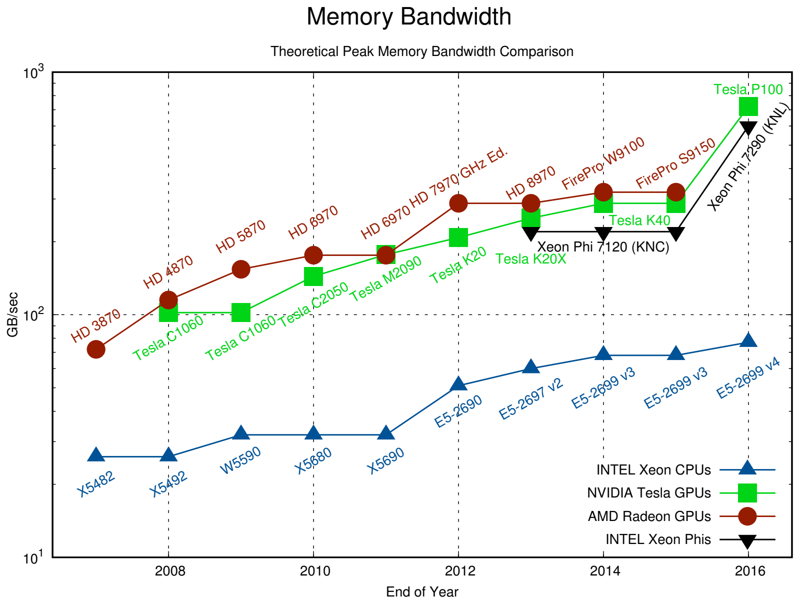
https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/

Introduction



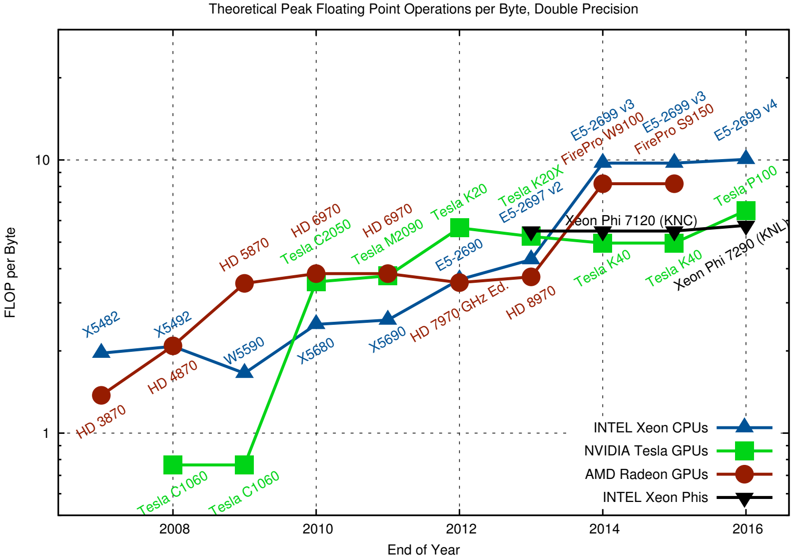
https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/

Introduction



https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/

Theoretical Peak Performance (FLOPs) per Byte of Memory Bandwidth



https://www.karlrupp.net/2013/06/cpu-gpu-and-mic-hardware-characteristics-over-time/

FLOPs and Bandwidth

Typical PETSc Operations

- Vector operations (add, dot, etc.)
- Sparse matrix-vector products (Krylov solvers, smoothers, residuals, etc.)

Maximizing Memory Bandwidth

- Read contiguous blocks of memory (contiguous access)
- Avoid unordered reads whenever possible

Check Memory Bandwidth Yourself

`make streams`

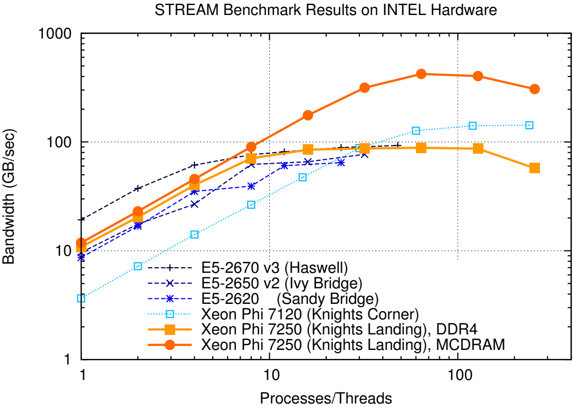
Performance question to `petsc-maint`, about 2h ago:

```
np  speedup
1   1.0
2   1.85
3   2.25
4   2.37
(...)
39  2.47
40  2.45
Estimation of possible speedup of MPI programs based on
    Streams benchmark.
It appears you have 1 node(s)
```

How does Memory Bandwidth Scale with Cores?

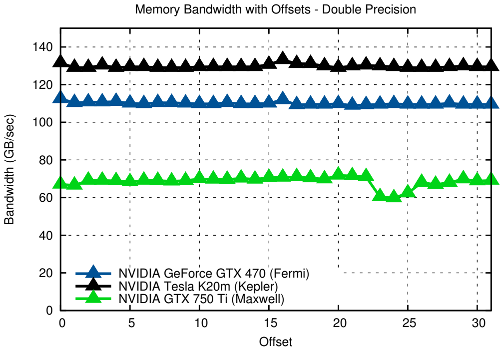
Usually saturates quickly

8-16 processes/threads usually suffice



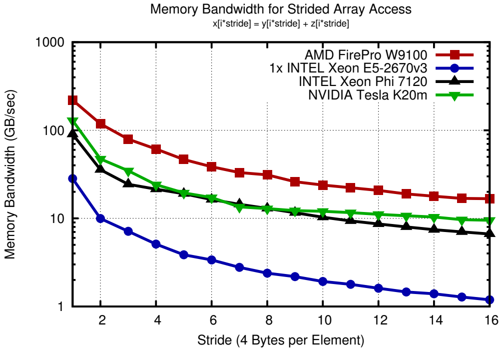
Offset Memory Access

```
void work(double *x, double *y, double *z, int N, int k)
{
  for (size_t i=0; i<N; ++i)
    z[i+k] = x[i+k] + y[i+k];
}
```



Strided Memory Access

```
void work(double *x, double *y, double *z, int N, int k)
{
  for (size_t i=0; i<N; ++i)
    z[i*k] = x[i*k] + y[i*k];
}
```

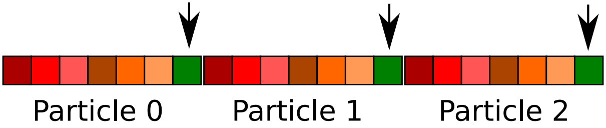


Strided Memory Access

Array of structs problematic

```
typedef struct particle
{
  double pos_x; double pos_y; double pos_z;
  double vel_x; double vel_y; double vel_z;
  double mass;
} Particle;

void increase_mass(Particle *particles, int N)
{
  for (int i=0; i<N; ++i)
    particles[i].mass *= 2.0;
}
```

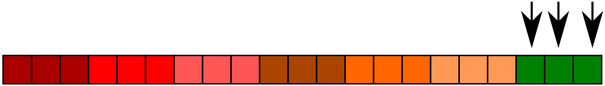


Strided Memory Access

Workaround: Structure of Arrays

```
typedef struct particles
{
  double *pos_x; double *pos_y; double *pos_z;
  double *vel_x; double *vel_y; double *vel_z;
  double *mass;
} Particle;

void increase_mass(Particle *particles, int N)
{
  for (int i=0; i<N; ++i)
    particles.mass[i] *= 2.0;
}
```



Mindlessly applying object-oriented programming

all the way down to fine granularity

is a recipe for a performance disaster.

Performance Modeling

Performance Modeling Mantra

At any given time during a run,
at least one hardware component
is operating at 100 percent capacity.

Bottleneck Potpourri

Latency

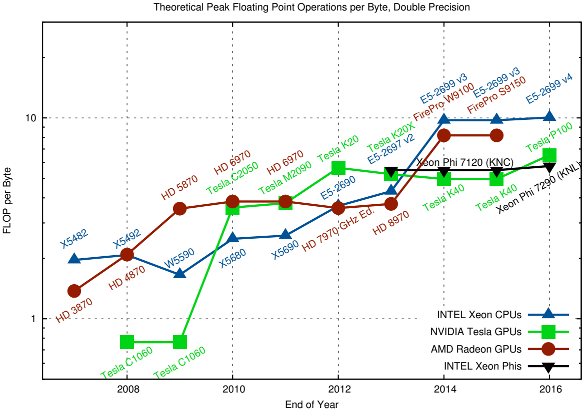
- Bottleneck in strong scaling limit
- Ultimate limit for time stepping

Latency - Sources

- Network latency (Ethernet $\sim 20\mu$s, Infiniband $\sim 5\mu$s)
- PCI-Express latency (Kernel launches, $\sim 10\mu$s)
- Thread synchronization (barriers, locks, $\sim 1 - 100\mu$s)
- Memory latency (~ 100ns)

Arithmetic Intensity

- Number of FLOPs per Byte
- FLOP-limited: Arithmetic intensity larger than $\sim$10
- Memory-limited: Arithmetic intensity smaller than $\sim$1

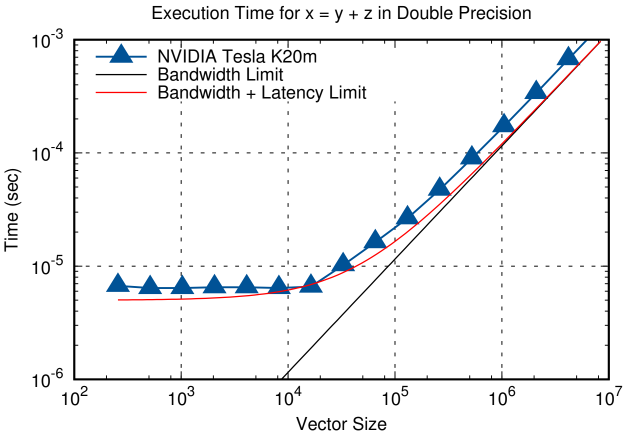


Vector Addition

$x = y + z$ with N elements each

1 FLOP per 24 byte in double precision

Limited by memory bandwidth $\Rightarrow T_2(N) \overset{?}{\approx} 3 \times 8 \times N/\text{Bandwidth} + \text{Latency}$

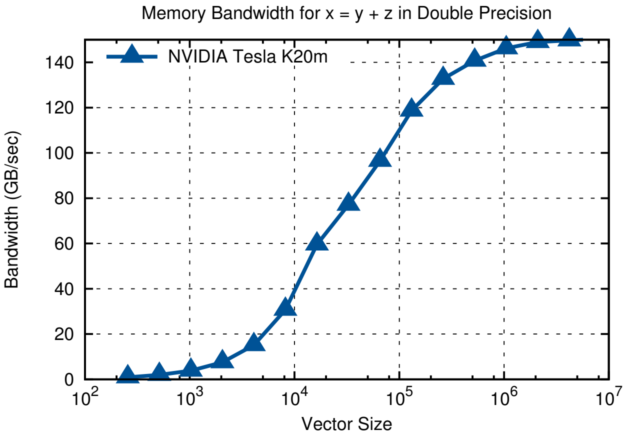


Vector Addition

$x = y + z$ with N elements each

1 FLOP per 24 byte in double precision

Limited by memory bandwidth $\Rightarrow T_2(N) \overset{?}{\approx} 3 \times 8 \times N/\text{Bandwidth} + \text{Latency}$



PETSc Profiling

PETSc Profiling

First: Get the Math Right!

- Choose an algorithm that gives robust iteration counts
- Choose an algorithm that really converges

Profiling

- Use `-log_view` for a performance profile
  - Event timing
  - Event flops
  - Memory usage
  - MPI messages
- Call `PetscLogStagePush()` and `PetscLogStagePop()`
  - User can add new stages
- Call `PetscLogEventBegin()` and `PetscLogEventEnd()`
  - User can add new events
- Call `PetscLogFlops()` to include your flops

PETSc Profiling

Reading -log_view

```
                         Max       Max/Min        Avg      Total
Time (sec):           1.548e+02     1.00122   1.547e+02
Objects:              1.028e+03     1.00000   1.028e+03
Flops:                1.519e+10     1.01953   1.505e+10  1.204e+11
Flops/sec:            9.814e+07     1.01829   9.727e+07  7.782e+08
MPI Messages:         8.854e+03     1.00556   8.819e+03  7.055e+04
MPI Message Lengths:  1.936e+08     1.00950   2.185e+04  1.541e+09
MPI Reductions:       2.799e+03     1.00000
```

Also a summary per stage

Memory usage per stage (based on when it was allocated)

Time, messages, reductions, balance, flops per event per stage

Always send `-log_view` when asking performance questions on mailing list

PETSc Profiling

```
Event                Count      Time (sec)     Flops                                             --- Global ---  --- Stage ---
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   Avg len Reduct  %T %F %M %L %R  %T %F %M %L %R
------------------------------------------------------------------------------------------------------------------------
--- Event Stage 1: Full solve
VecDot                43 1.0 4.8879e-02 8.3 1.77e+06 1.0 0.0e+00 0.0e+00 4.3e+01  0  0  0  0  0   0  0  0  0  1
VecMDot             1747 1.0 1.3021e+00 4.6 8.16e+07 1.0 0.0e+00 0.0e+00 1.7e+03  0  1  0  0 14   1  1  0  0 27
VecNorm             3972 1.0 1.5460e+00 2.5 8.48e+07 1.0 0.0e+00 0.0e+00 4.0e+03  0  1  0  0 31   1  1  0  0 61
VecScale            3261 1.0 1.6703e-01 1.0 3.38e+07 1.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0
VecScatterBegin     4503 1.0 4.0440e-01 1.0 0.00e+00 0.0 6.1e+07 2.0e+03 0.0e+00  0  0 50 26  0   0  0 96 53  0
VecScatterEnd       4503 1.0 2.8207e+00 6.4 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0
MatMult             3001 1.0 3.2634e+01 1.1 3.68e+09 1.1 4.9e+07 2.3e+03 0.0e+00 11 22 40 24  0  22 44 78 49  0
MatMultAdd           604 1.0 6.0195e-01 1.0 5.66e+07 1.0 3.7e+06 1.3e+02 0.0e+00  0  0  3  0  0   0  1  6  0  0
MatMultTranspose     676 1.0 1.3220e+00 1.6 6.50e+07 1.0 4.2e+06 1.4e+02 0.0e+00  0  0  3  0  0   1  1  7  0  0
MatSolve            3020 1.0 2.5957e+01 1.0 3.25e+09 1.0 0.0e+00 0.0e+00 0.0e+00  9 21  0  0  0  18 41  0  0  0
MatCholFctrSym         3 1.0 2.8324e-04 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 0.0e+00  0  0  0  0  0   0  0  0  0  0
MatCholFctrNum        69 1.0 5.7241e+00 1.0 6.75e+08 1.0 0.0e+00 0.0e+00 0.0e+00  2  4  0  0  0   4  9  0  0  0
MatAssemblyBegin     119 1.0 2.8250e+00 1.5 0.00e+00 0.0 2.1e+06 5.4e+04 3.1e+02  1  0  2 24  2   2  0  3 47  5
MatAssemblyEnd       119 1.0 1.9689e+00 1.4 0.00e+00 0.0 2.8e+05 1.3e+03 6.8e+01  1  0  0  0  1   1  0  0  0  1
SNESSolve              4 1.0 1.4302e+02 1.0 8.11e+09 1.0 6.3e+07 3.8e+03 6.3e+03 51 50 52 50 50  99100 99100 97
SNESLineSearch        43 1.0 1.5116e+01 1.0 1.05e+08 1.1 2.4e+06 3.6e+03 1.8e+02  5  1  2  2  1  10  1  4  4  3
SNESFunctionEval      55 1.0 1.4930e+01 1.0 0.00e+00 0.0 1.8e+06 3.3e+03 8.0e+00  5  0  1  1  0  10  0  3  3  0
SNESJacobianEval      43 1.0 3.7077e+01 1.0 7.77e+06 1.0 4.3e+06 2.6e+04 3.0e+02 13  0  4 24  2  26  0  7 48  5
KSPGMRESOrthog      1747 1.0 1.5737e+00 2.9 1.63e+08 1.0 0.0e+00 0.0e+00 1.7e+03  1  1  0  0 14   1  2  0  0 27
KSPSetup             224 1.0 2.1040e-02 1.0 0.00e+00 0.0 0.0e+00 0.0e+00 3.0e+01  0  0  0  0  0   0  0  0  0  0
KSPSolve              43 1.0 8.9988e+01 1.0 7.99e+09 1.0 5.6e+07 2.0e+03 5.8e+03 32 49 46 24 46  62 99 88 48 88
PCSetUp              112 1.0 1.7354e+01 1.0 6.75e+08 1.0 0.0e+00 0.0e+00 8.7e+01  6  4  0  0  1  12  9  0  0  1
PCSetUpOnBlocks     1208 1.0 5.8182e+00 1.0 6.75e+08 1.0 0.0e+00 0.0e+00 8.7e+01  2  4  0  0  1   4  9  0  0  1
PCApply              276 1.0 7.1497e+01 1.0 7.14e+09 1.0 5.2e+07 1.8e+03 5.1e+03 25 44 42 20 41  49 88 81 39 79
```

Communication Costs

- Reductions: usually part of Krylov method, latency limited
  - `VecDot`
  - `VecMDot`
  - `VecNorm`
  - `MatAssemblyBegin`
  - Change algorithm (e.g. IBCGS)
- Point-to-point (nearest neighbor), latency or bandwidth
  - `VecScatter`
  - `MatMult`
  - `PCApply`
  - `MatAssembly`
  - `SNESFunctionEval`
  - `SNESJacobianEval`
  - Compute subdomain boundary fluxes redundantly
  - Ghost exchange for all fields at once
  - Better partition

Adding a Logging Event (C)

```
PetscLogEvent  USER_EVENT;
PetscClassId   classid;
PetscLogDouble user_event_flops;

PetscClassIdRegister("class name",&classid);
PetscLogEventRegister("user event",classid,&USER_EVENT);

PetscLogEventBegin(USER_EVENT,0,0,0,0);
   /* code segment to monitor */
PetscLogFlops(user_event_flops);
PetscLogEventEnd(USER_EVENT,0,0,0,0);
```

Adding a Logging Event (Python)

```
with PETSc.logEvent('Reconstruction') as recEvent:
    # All operations are timed in recEvent
    reconstruct(sol)
    # Flops are logged to recEvent
    PETSc.Log.logFlops(user_event_flops)
```

Adding a Logging Stage (C)

```
PetscLogStage stage;

PetscLogStageRegister("name", &stage);
PetscLogStagePush(stage);

/* Code to Monitor */

PetscLogStagePop();
```

PETSc Profiling

```
Event                Count      Time (sec)     Flops                             --- Global ---  --- Stage --
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   Avg len Reduct  %T %F %M %L %R  %T %F %M %L
------------------------------------------------------------------------------------------------------------------
--- Event Stage 0: Main Stage

MatMult              178 1.0 7.8040e+01 1.0 2.59e+11 1.0 4.4e+02 2.0e+05 0.0e+00  33 41  6 11  0  51 89 20 24
MatPtAP               10 1.0 2.4870e+01 1.0 5.45e+09 1.0 2.1e+02 3.1e+05 1.8e+02  10  1  3  8  1  16  2  9 18
MatPtAPSymbolic       10 1.0 1.8828e+01 1.0 0.00e+00 0.0 1.2e+02 2.7e+05 8.2e+01   8  0  2  4  0  12  0  5  9
MatPtAPNumeric        10 1.0 6.0428e+00 1.0 5.45e+09 1.0 9.4e+01 3.7e+05 1.0e+02   3  1  1  4  0   4  2  4  9
SNESSolve              2 1.0 1.9059e+02 1.0 6.22e+11 1.0 6.6e+03 9.3e+04 3.4e+03  79 99 92 75 16 123213292168
KSPSolve               2 1.0 1.8230e+02 1.0 6.07e+11 1.0 6.5e+03 9.1e+04 3.2e+03  76 97 89 72 15 118208285161
PCSetUp                8 1.0 1.6138e+01 1.0 4.81e+09 1.1 1.2e+03 8.1e+04 2.5e+03   7  1 17 12 11  10  2 55 28
PCApply               46 1.0 1.2586e+02 1.0 4.43e+11 1.0 6.3e+03 8.5e+04 2.7e+03  52 70 87 65 12  81152277146
KSPSolve_FS_0         46 1.0 1.0038e+02 1.0 3.42e+11 1.0 6.2e+03 8.2e+04 2.6e+03  42 54 86 62 12  65117273138
(...)

--- Event Stage 1: MG Apply

MatMultMFA11         296 1.0 4.3461e+01 1.0 2.82e+11 1.0 1.2e+03 3.0e+05 0.0e+00  18 45 16 43  0  51 84 24 78
KSPSolve             230 1.0 7.2581e+01 1.0 2.87e+11 1.0 4.5e+03 8.5e+04 2.6e+02  30 46 62 47  1  85 85 91 841
PCApply              642 1.0 1.0269e+01 1.0 1.40e+10 1.1 3.0e+03 8.7e+03 1.8e+02   4  2 42  3  1  12  4 61  6
MGSmooth Level 0      46 1.0 7.8169e+00 1.0 1.06e+10 1.1 3.0e+03 8.3e+03 1.7e+02   3  2 41  3  1   9  3 61  5
MGSmooth Level 1      92 1.0 2.4177e+01 1.0 3.17e+10 1.0 5.0e+02 1.2e+05 4.6e+01  10  5  7  7  0  28  9 10 13
MGResid Level 1       46 1.0 4.3231e+00 1.0 5.77e+09 1.0 9.2e+01 1.2e+05 0.0e+00   2  1  1  1  0   5  2  2  2
MGInterp Level 1      92 1.0 3.5063e-01 1.1 1.09e+08 1.0 9.2e+01 1.5e+04 0.0e+00   0  0  1  0  0   0  0  2  0
MGSmooth Level 2      92 1.0 4.0886e+01 1.0 2.44e+11 1.0 1.0e+03 3.0e+05 4.6e+01  17 39 14 37  0  48 73 20 66
MGResid Level 2       46 1.0 6.8277e+00 1.0 4.39e+10 1.0 1.8e+02 3.0e+05 0.0e+00   3  7  3  7  0   8 13  4 12
MGInterp Level 2      92 1.0 1.0898e+00 1.4 8.47e+08 1.0 9.2e+01 3.8e+04 0.0e+00   0  0  1  0  0   1  0  2  1
(...)
------------------------------------------------------------------------------------------------------------------
```

PETSc and Threads

PETSc and Threads

Competing Threading Approaches

- pthread
- OpenMP
- C++11 threads
- Compiler magic
- ...

Issues with Threads

- Problematic across compilers
- Data locality
- Thread ownership
- Software interface

Assumption for Subsequent Discussion

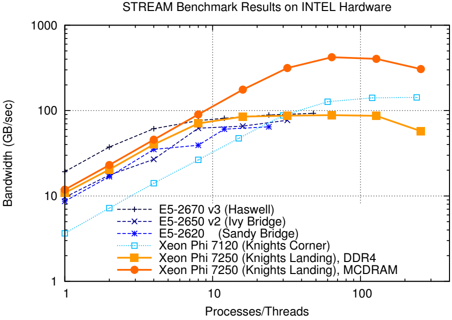
- Primary Goal: Get the science done!
- 10 percent performance difference is **not significant**

When to Use Threads After All?

Distributed Memory: Need MPI anyway

Shared Memory: MPI for multi-socket systems for NUMA reasons

1-Socket Machines: If a 2-5x gain is critical, use a cluster!




```
numactl --membind 0 ./stream # DDR4
numactl --membind 1 ./stream # MCDRAM
```

Threads and Library Interfaces

Attempt 1

Library spawns threads

```
void library_func(double *x, int N) {
  #pragma omp parallel for
  for (int i=0; i<N; ++i) x[i] = something_complicated();
}
```

Problems

Call from multi-threaded environment?

```
void user_func(double **y, int N) {
  #pragma omp parallel for
  for (int j=0; j<M; ++j) library_func(y[j], N);
}
```

Incompatible OpenMP runtimes (e.g. GCC vs. ICC)

Attempt 2

Use pthreads/TBB/etc. instead of OpenMP to spawn threads

Fixes incompatible OpenMP implementations (probably)

Problems

Still a problem with multi-threaded user environments

```
void user_func(double **y, int N) {
  #pragma omp parallel for
  for (int j=0; j<M; ++j) library_func(y[j], N);
}
```

Threads and Library Interfaces

Attempt 3

Hand back thread management to user

```
void library_func(ThreadInfo ti, double *x, int N) {
  int start = compute_start_index(ti, N);
  int stop  = compute_stop_index(ti, N);
  for (int i=start; i<stop; ++i)
    x[i] = something_complicated();
}
```

Implications

Users can use their favorite threading model

API requires one extra parameter

Extra boilerplate code required in user code

Threads and Library Interfaces

Reflection

Extra thread communication parameter

```
void library_func(ThreadInfo ti, double *x, int N) {...}
```

Rename thread management parameter

```
void library_func(Thread_Comm c, double *x, int N) {...}
```

Compare:

```
void library_func(MPI_Comm comm, double *x, int N) {...}
```

Conclusion

Prefer flat MPI over MPI+OpenMP for a composable software stack

MPI automatically brings better data locality

PETSc and GPUs

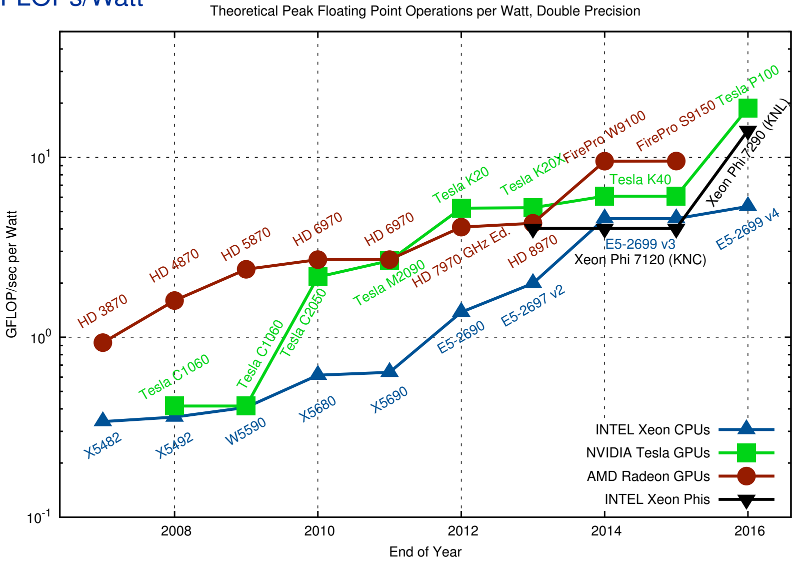
Why bother?

Don't believe anything
unless you can run it

Matt Knepley

Why bother?

GFLOPs/Watt



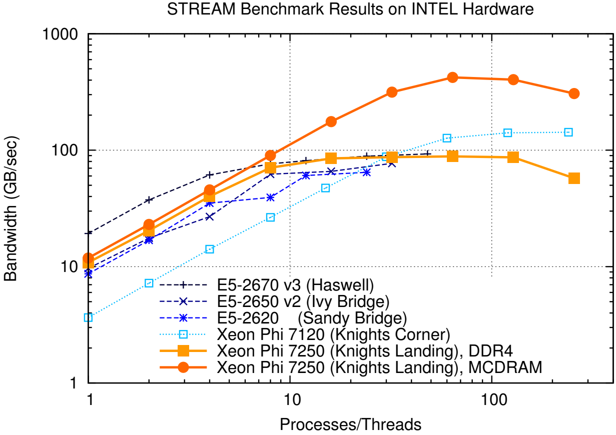
Why bother?

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



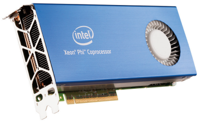
PETSc on GPUs and MIC:

Current Status

Available Options

Native on Xeon Phi

Cross-compile for Xeon Phi



CUDA

CUDA-support through CUSP

`-vec_type cusp -mat_type aijcusp`

`-vec_type cuda -mat_type aijcusparse`

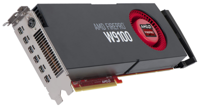
Only for NVIDIA GPUs

CUDA/OpenCL/OpenMP

CUDA/OpenCL/OpenMP-support through ViennaCL

`-vec_type viennacl -mat_type aijviennacl`

OpenCL on CPUs and MIC fairly poor



Configuration

CUDA (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_60
```

OpenCL (ViennaCL)

OpenCL-enabled configuration

```
./configure [..] --download-viennacl
   --with-opencl-include=/path/to/OpenCL/include
   --with-opencl-lib=/path/to/libOpenCL.so
```

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

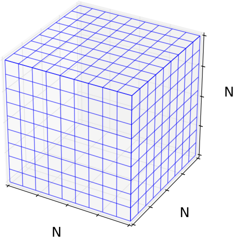
Pitfall: Function Pointers

- Pass CUDA function “pointers” through library boundaries?
- OpenCL: Pass kernel sources, user-data hard to pass
- Composability?

Pitfall: 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 (!!)**

Many PETSc Operations

do NOT benefit from modern high-end GPUs

in a substantial way!

(Exception: OpenPower systems)

Current GPU-Functionality in PETSc

Current GPU-Functionality in PETSc

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

Additional Functionality

MatMult via cuSPARSE

OpenCL residual evaluation for PetscFE

PETSc on GPUs and MIC:

Current Directions

Current: CUDA

Split CUDA-buffers from CUSP

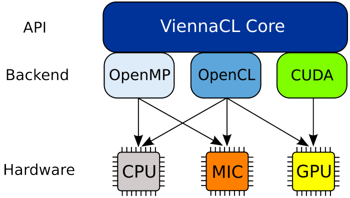
- Vector operations by cuBLAS
- MatMult by different packages
- CUSP (and others) provides add-on functionality

More CUSP Functionality in PETSc

- Relaxations (Gauss-Seidel, SOR)
- Polynomial preconditioners
- Approximate inverses

ViennaCL

- CUDA, OpenCL, OpenMP backends
- Backend switch at **runtime**
- CUDA, OpenCL and OpenMP 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

Current: PETSc + ViennaCL

Current 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

GPU Summary and Conclusion

Currently Available

- CUSP/CUDA for CUDA, ViennaCL for CUDA/OpenCL/OpenMP
- Automatic use for vector operations and SpMV
- Smoothed Agg. AMG via CUSP and ViennaCL
- ViennaCL as CUDA/OpenCL/OpenMP-hydra

Current Activities

- GPU-acceleration for GAMG
- Better support for $n > 1$ processes
- Use of cuBLAS and cuSPARSE

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