FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Performance Engineering of the Kernel Polynomial Method on Large-Scale CPU-GPU Systems

Moritz Kreutzer*, Georg Hager*, Gerhard Wellein*,
Andreas Pieper#, Andreas Alvermann#, Holger Fehske#

*: Erlangen Regional Computing Center, Friedrich-Alexander University of Erlangen-Nuremberg
#: Institute of Physics, Ernst Moritz Arndt University of Greifswald

Friday Lunch Talk @ Innovative Computing Laboratory

November 06, 2015

The University of Tennessee, Knoxville, TN, USA

This work was supported (in part) by the German Research Foundation (DFG) through the Priority Programs 1648 “Software for Exascale Computing” under project ESSEX and 1459 “Graphene”.

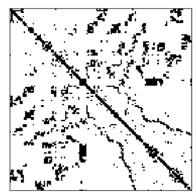
Prologue

What is this about?

The Kernel Polynomial Method (KPM)

Approximate the complete eigenvalue spectrum of a large sparse matrix.

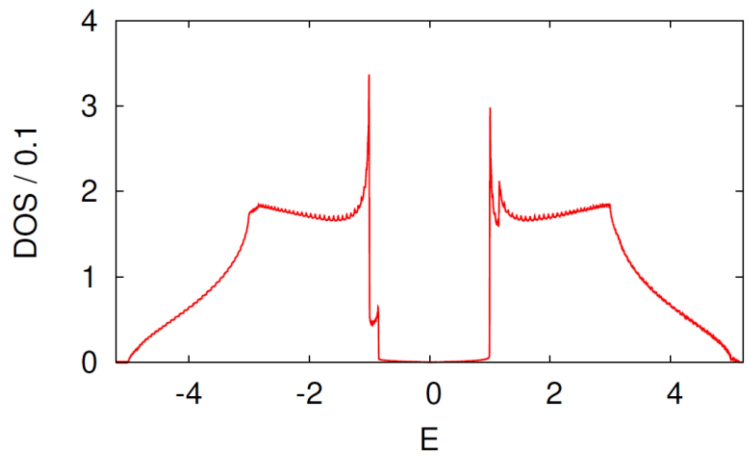
Large, Sparse



$$\boldsymbol{H}\,\boldsymbol{x} = \lambda\,\boldsymbol{x}$$

$$\underbrace{\{\lambda_1, \lambda_2, \dots, \lambda_k, \dots, \lambda_{n-1}, \lambda_n\}}$$

Good approximation to full spectrum
(e.g. Density of States)

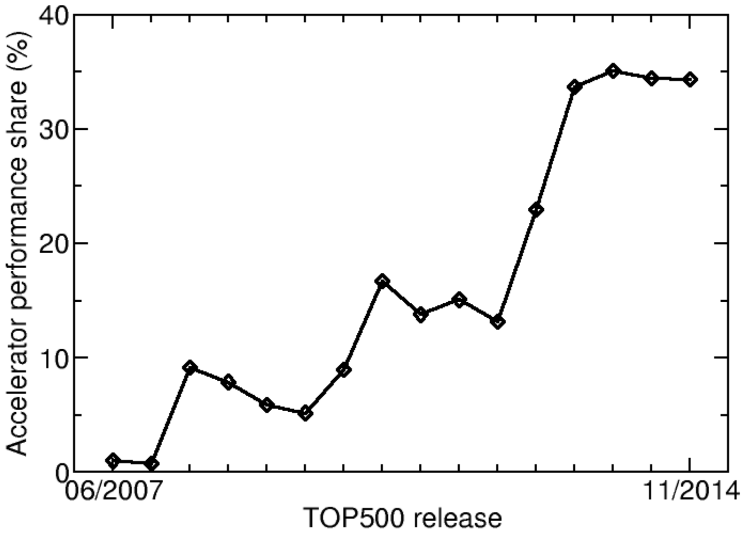


A. Weiße, G. Wellein, A. Alvermann, H. Fehske: **"The kernel polynomial method"**, Rev. Mod. Phys., vol. 78, p. 275, 2006.
E. di Napoli, E. Polizzi, Y. Saad: **"Efficient estimation of eigenvalue counts in an interval"**, Preprint. http://arxiv.org/abs/1308.4275

Why optimize for heterogeneous systems?

One third of TOP500 performance stems from accelerators.

But: Few <u>truly</u> heterogeneous software.
(Using both CPUs and accelerators.)



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

The Kernel Polynomial Method

Algorithmic Analysis

The Kernel Polynomial Method

Compute Chebyshev polynomials and moments.

Basic algorithm and algorithmic optimizations:

Exploit knowledge from all software layers!

Building blocks: (Sparse) linear algebra library

for $r = 0$ to $R-1$ **do** — Application: Loop over random initial states
  $|v\rangle \leftarrow |\mathrm{rand}()\rangle$
  Initialization steps and computation of η_0, η_1
  for $m = 1$ to $M/2$ **do** — Algorithm: Loop over moments
    $\mathrm{swap}(|w\rangle, |v\rangle)$
    $|u\rangle \leftarrow H|v\rangle$ ▷ `spmv()` Sparse matrix vector multiply
    $|u\rangle \leftarrow |u\rangle - b|v\rangle$ ▷ `axpy()` Scaled vector addition
    $|w\rangle \leftarrow -|w\rangle$ ▷ `scal()` Vector scale
    $|w\rangle \leftarrow |w\rangle + 2a|u\rangle$ ▷ `axpy()` Scaled vector addition
    $\eta_{2m} \leftarrow \langle v|v\rangle$ ▷ `nrm2()` Vector norm
    $\eta_{2m+1} \leftarrow \langle w|v\rangle$ ▷ `dot()` Dot Product
  end for
end for

The Kernel Polynomial Method

Compute Chebyshev polynomials and moments.

Basic algorithm and algorithmic optimizations:

Exploit knowledge from all software layers!

```
for r = 0 to R−1 do
    |v⟩ ← |rand()⟩
    Initialization steps and computation of η0, η1
    for m = 1 to M/2 do
        swap(|w⟩, |v⟩)
        |u⟩      ← H|v⟩              ▷ spmv()
        |u⟩      ← |u⟩ − b|v⟩        ▷ axpy()
        |w⟩      ← −|w⟩              ▷ scal()
        |w⟩      ← |w⟩ + 2a|u⟩       ▷ axpy()
        η2m      ← ⟨v|v⟩             ▷ nrm2()
        η2m+1    ← ⟨w|v⟩             ▷ dot()
    end for
end for
```

```
for r = 0 to R−1 do
    |v⟩ ← |rand()⟩
    Initialization steps and computation of η0, η1
    for m = 1 to M/2 do
        swap(|w⟩, |v⟩)
        |w⟩ = 2a(H − b𝟙)|v⟩ − |w⟩ &
            η2m   = ⟨v|v⟩ &
            η2m+1 = ⟨w|v⟩            ▷ aug_spmv()
end for
```

Augmented Sparse Matrix Vector Multiply

The Kernel Polynomial Method

Compute Chebyshev polynomials and moments.

Basic algorithm and algorithmic optimizations:

Exploit knowledge from all software layers!

for $r = 0$ to $R-1$ **do**
  $|v\rangle \leftarrow |\text{rand}()\rangle$
  Initialization steps and computation of η_0, η_1
  for $m = 1$ to $M/2$ **do**
    $\text{swap}(|w\rangle, |v\rangle)$
    $|u\rangle \leftarrow H|v\rangle$ ▷ `spmv()`
    $|u\rangle \leftarrow |u\rangle - b|v\rangle$ ▷ `axpy()`
    $|w\rangle \leftarrow -|w\rangle$ ▷ `scal()`
    $|w\rangle \leftarrow |w\rangle + 2a|u\rangle$ ▷ `axpy()`
    $\eta_{2m} \leftarrow \langle v|v\rangle$ ▷ `nrm2()`
    $\eta_{2m+1} \leftarrow \langle w|v\rangle$ ▷ `dot()`
  end for
end for

for $r = 0$ to $R-1$ **do**
  $|v\rangle \leftarrow |\text{rand}()\rangle$
  Initialization steps and computation of η_0, η_1
  for $m = 1$ to $M/2$ **do**
    $\text{swap}(|w\rangle, |v\rangle)$
    $|w\rangle = 2a(H - b\mathbb{1})|v\rangle - |w\rangle$ &
    $\eta_{2m} = \langle v|v\rangle$ &
    $\eta_{2m+1} = \langle w|v\rangle$ ▷ `aug_spmv()`
  end for

$|V\rangle := |v\rangle_{0..R-1}$ ▷ Assemble vector blocks
$|W\rangle := |w\rangle_{0..R-1}$
$|V\rangle \leftarrow |\text{rand}()\rangle$
Initialization steps and computation of μ_0, μ_1
for $m = 1$ to $M/2$ **do**
  $\text{swap}(|W\rangle, |V\rangle)$
  $|W\rangle = 2a(H - b\mathbb{1})|V\rangle - |W\rangle$ &
  $\eta_{2m}[:] = \langle V|V\rangle$ &
  $\eta_{2m+1}[:] = \langle W|V\rangle$ ▷ `aug_spmmv()`
end for

Augmented Sparse Matrix Multiple Vector Multiply

Analysis of the Algorithmic Optimization

- **Minimum code balance of vanilla algorithm:**
  complex double precision values, 32-bit indices, 13 non-zeros per row, application: topological insulators

  $B_{vanilla} = 3.39\ Bytes/Flop$ (B = inverse computational intensity)

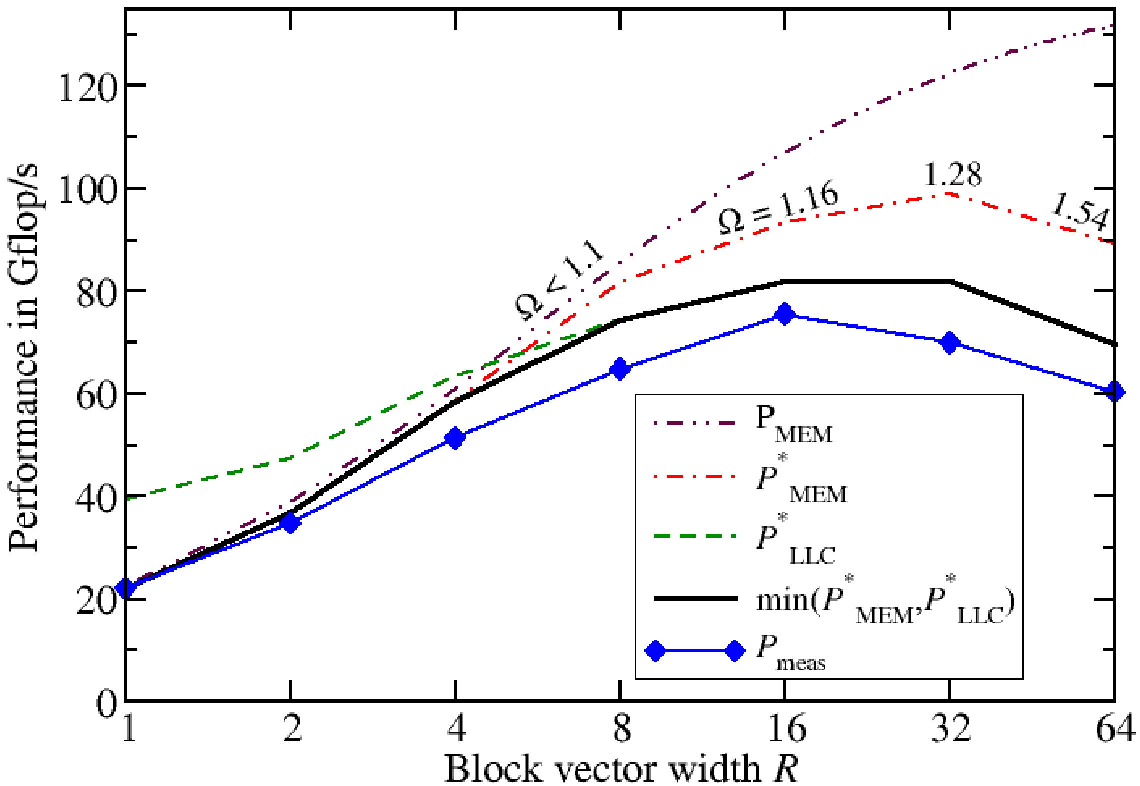
- **Identified bottleneck: Memory bandwidth**
  ➔ Decrease memory transfers to alleviate bottleneck

- **Algorithmic optimizations reduce code balance:**
  $B_{aug\_spmv} = 2.23\ B/F$ kernel fusion
  $B_{aug\_spmmv}(R) = 1.88/R + 0.35\ B/F$ put *R* vectors in block

Consequences of Algorithmic Optimization

- **Mitigation of the relevant bottleneck**
  ➔ Expected speedup ☺
- **Other bottlenecks become relevant**
  ➔ Achieved speedup may not be $B_{vanilla}/B_{aug\_spmmv}$ 😐
- **Block vectors are best stored interleaved**
  ➔ May impose larger changes to the codebase ☹
- `aug_spmmv()` **no part of standard libraries**
  ➔ Implementation by hand is necessary ☹

CPU roofline performance model



$P = \frac{b}{B}$ **Gflop/s**

→ Performance limit for bandwidth-bound code

b = max. bandwidth = 50 GB/s
B = code balance

$\Omega = \frac{\textit{Actual data transfers}}{\textit{Minimum data transfers}}$

Intel Xeon E5-2660v2 "Ivy Bridge"

S. Williams, A. Waterman, D. Patterson: **"Roofline: An insightful visual performance model for multicore architectures"**, *Commun. ACM*, vol. 52, p. 65, 2009.

Implementation

How to harness a heterogeneous machine in an efficient way?

Implementation

Algorithmic optimizations lead to a *potential* speedup.

➔ We "merely" need an efficient implementation!

Data or task parallelism?

- **MAGMA: task parallelism between devices**
  ➔ Kernel fusion ⚡ Task parallelism

➔ Data-parallel approach suits our needs

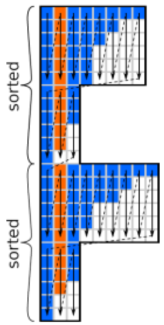
Implementation

Data-parallel heterogeneous work distribution

- Static work-distribution by matrix rows/entries
- Device workload ←→ device performance

SELL-C-σ sparse matrix storage format

- Unified format for all relevant devices
- Currently no runtime-exchange of matrix data (dynamic load balancing, future work)

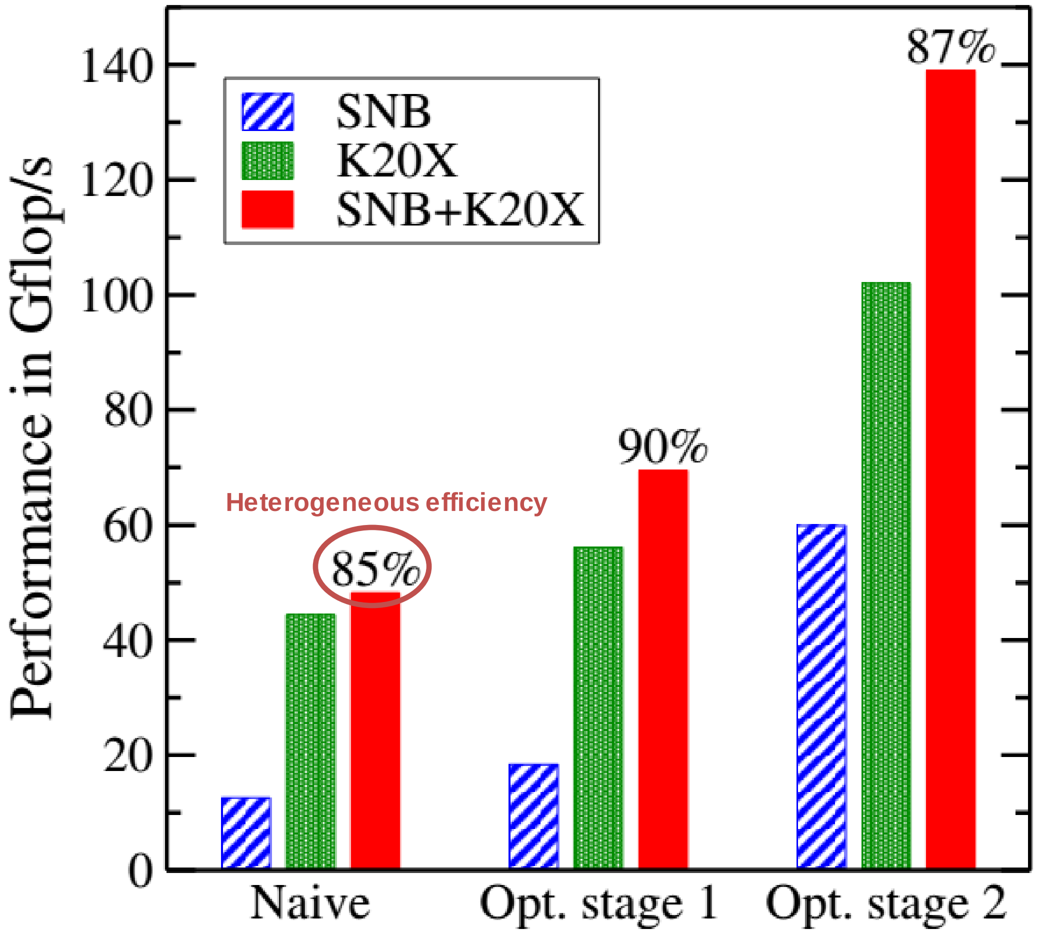


M. Kreutzer, G. Hager, G. Wellein, H. Fehske, A. R. Bishop, **"A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units"**, *SIAM J. Sci. Comput.*, vol. 36, p. C401, 2014

Performance results

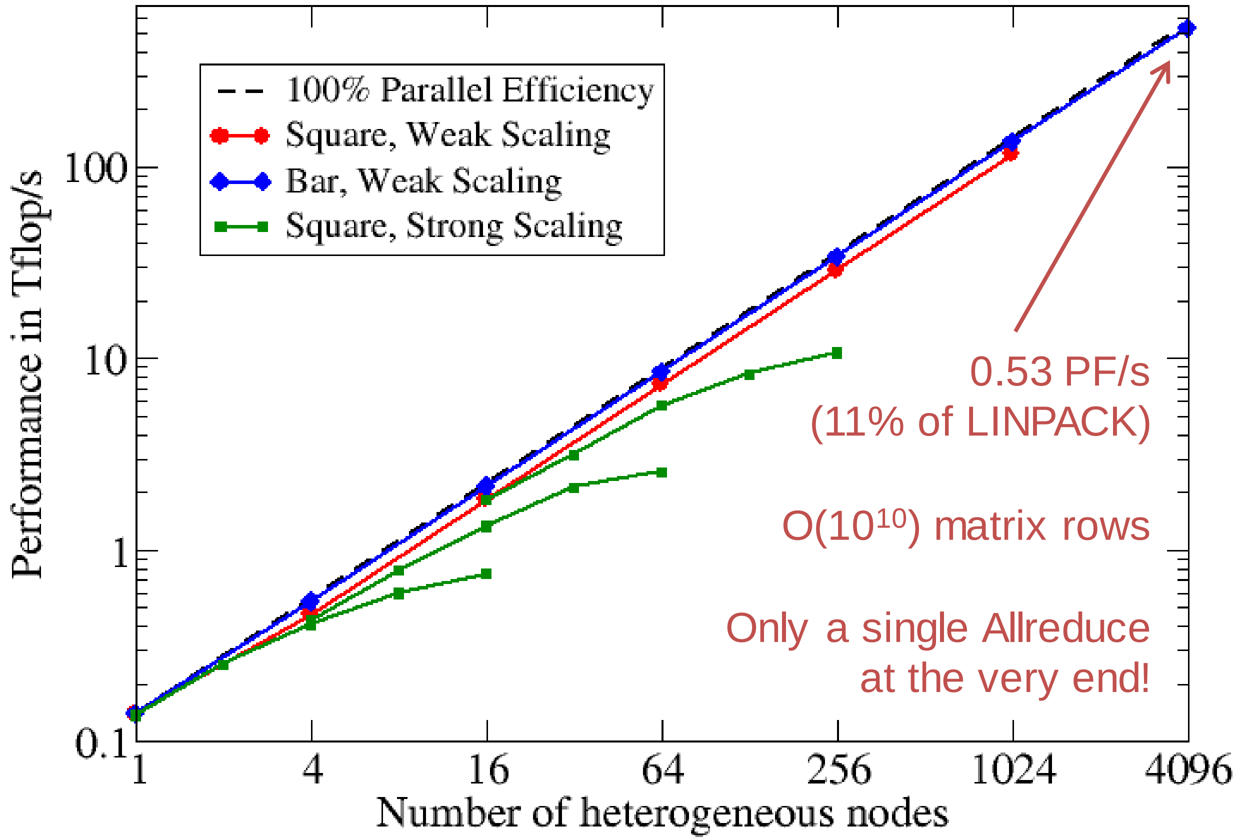
Does all this really pay off?

Single-node Heterogeneous Performance



SNB: Intel Xeon E5-2670 “Sandy Bridge“, K20X: Nvidia Tesla K20X, Complex double precision data (topological insulator)

Large-scale Heterogeneous Performance



CRAY XC30 – Piz Daint*

- 5272 nodes, each w/
  - 1 octacore Intel Sandy Bridge
  - 1 Nvidia Kepler K20x
- Peak: 7.8 Pflop/s
- LINPACK: 6.3 Pflop/s
- Largest system in Europe

*Thanks to CSCS/O. Schenk/T. Schulthess for granting access and compute time

M. Kreutzer, A.Pieper, G.Hager, G. Wellein, A. Alvermann, H. Fehske, **"Performance Engineering of the Kernel Polynomial Method on Large-Scale CPU-GPU Systems"**, *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, p. 417-426, 2015

Epilogue

Try it out! (If you want…)

Download our building block library and KPM application: `http://tiny.cc/ghost`

General, Hybrid, and Optimized Sparse Toolkit

- **MPI + OpenMP + SIMD + CUDA**
- **Transparent data-parallel heterogeneous execution**
- **Affinity-aware task parallelism (checkpointing, comm. hiding, etc.)**
- **Support for block vectors**
  - Automatic code generation for common block vector sizes
  - Hand-implemented tall skinny dense matrix kernels
- **Fused kernels (arbitrarily "augmented SpMMV")**
- **SELL-C-σ heterogeneous sparse matrix format**
- **Various sparse eigensolvers implemented and downloadable**

. . .