Projects

High-performance matrix computations

pauldj@aices.rwth-aachen.de

- For all projects:
  - The code needs to be properly working
    $\Rightarrow$ prepare a small demo
  - Correctness first, then performance;
    convince me that the code is correct
  - Prepare figures and a document to present the results
- Exam dates:
  - before Friday, July 26
  - between August 15 and August 30
  - between October 2 and October 13

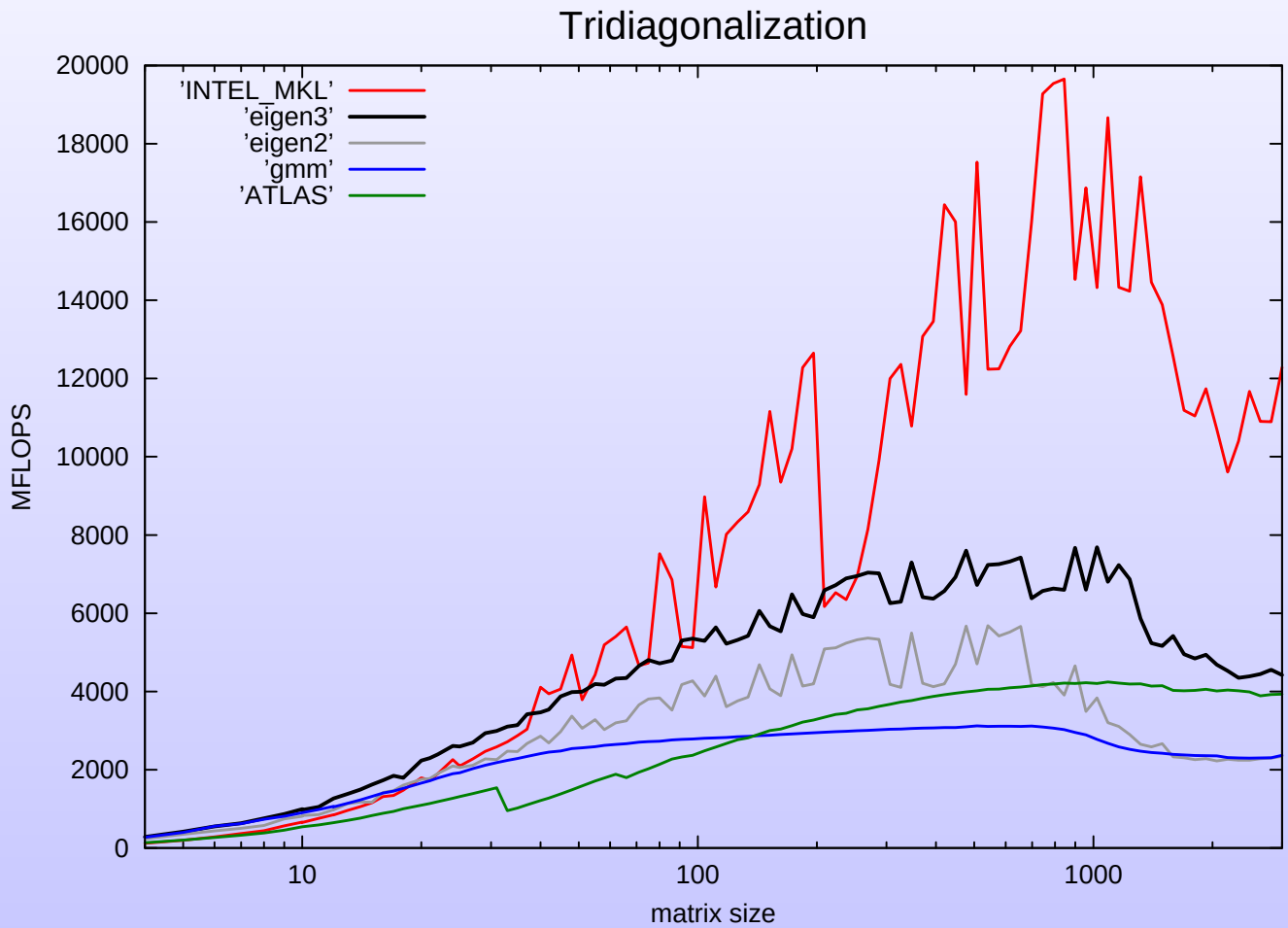
[N] SMP – Small eigenproblems

- Input: thousands of small (tridiagonal) eigenproblems ($5 \leq n \lesssim 60$).
- Objective: investigate/compare the performance and the accuracy of MR3-SMP and LAPACK's solvers.
- C, C++, or Fortran
- LAPACK: MKL or `www.netlib.org/lapack/`
  MR3-SMP: `http://code.google.com/p/mr3smp/`
- Consider BX+InvIt, QR, DC, MR3, and MR3-SMP.
- Accuracy:
  relative residual $\dfrac{\|TX - X\Lambda\|}{\|T\|}$, orthogonality $\|X^T X - I\|$
- Test different types of spectrum! $\rightarrow$ ask me for details
- What is the best way to utilize multiple cores?

[N] SMP – SBR

- Implement the “Successive Band Reduction” on a multi-core architecture.
- Matrices of size $100 \lesssim n \lesssim 10000$
- C, C++, or Fortran
- Correctness: make sure $\lambda(A) = \lambda(T)$
- Performance: what’s the best bandwidth?
- Performance: how does SBR compare with the blocked reduction?
- Performance: how does SBR scale with the number of cores?
- *“A Framework for Symmetric Band Reduction”*,
  Christian Bischof, Bruno Lang, Xiaobai Sun. 1999.
  `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.6529`

[N] SMP – Tridiagonalization in Eigen (1/2)



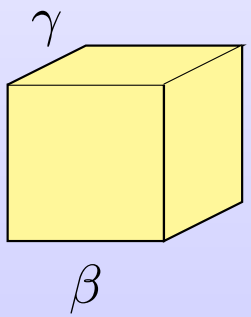
From `eigen.tuxfamily.org/`

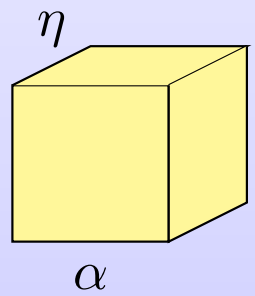
[N] SMP – Tridiagonalization in Eigen (2/2)

- `eigen.tuxfamily.org/`
- What algorithm does Eigen implement? Is it blocked? Is it parallel? Differences with respect to “our” blocked algorithm?
- Performance: how does Eigen compare to LAPACK?
- Performance: how does Eigen scale with the number of cores?
- Where is the bottleneck?
- Can you improve it? (careful: this might be quite challenging)

[N] MPI – Tensors contractions (1/2)

- A and $B \in \mathbb{R}^{n \times n \times n}$, $R \in \mathbb{R}^{n \times n}$, $\quad n \lesssim 250$.
- Two nodes: A and B are stored in P_0 and P_1, respectively. Assume each node has enough memory to store only $n^3 + b * n^2$ doubles.
- Storage by columns and by face.

$A^{\alpha\beta\gamma}$ = 

$B_{\gamma\alpha\eta}$ = 

- 1) $R^{\beta}{}_{\eta} := A^{\alpha\beta\gamma} B_{\gamma\alpha\eta}$ (Einstein notation)

$$\forall_\beta \forall_\eta r_{\beta\eta} := \sum_\alpha \sum_\gamma a_{\alpha\beta\gamma} b_{\gamma\alpha\eta} \equiv \forall_j \forall_w r_{jw} := \sum_i \sum_k a_{ijk} b_{kiw}$$

[N] MPI – Tensors contractions (2/2)

- Compute R (on either P_0 or P_1, or on both of them).
- Use as much BLAS as possible; in particular, as much BLAS3 as possible. Time different approaches.
- `http://arxiv.org/pdf/1307.2100`
- 2) $R^{\alpha}{}_{\eta} := A^{\alpha\beta\gamma} B_{\eta\beta\gamma} \equiv \forall_i \forall_w r_{iw} := \sum_j \sum_k a_{ijk} b_{wjk}$

- Input: a sequence of cubes $\in \mathbb{R}^{n \times n \times n}$, stored in a file. $n \lesssim 250$.
- 8 nodes: $p_{000}, \ldots, p_{111}$.
  Assume that each node has enough memory to store only $3 * \frac{n^3}{8}$ doubles.
- For each cube, compute D (see below), and store it in a file.

- k-th cube:

  1) p_{ijk} loads the corresponding octant A_{ijk}

  2) $$\begin{array}{l|l} B_{000} := A_{000} + A_{001} & B_{001} := A_{000} - A_{001} \\ \hline B_{010} := A_{010} + A_{011} & B_{011} := A_{010} - A_{011} \end{array}$$, and
  $$\begin{array}{l|l} B_{100} := A_{100} + A_{101} & B_{101} := A_{100} - A_{101} \\ \hline B_{110} := A_{110} + A_{111} & B_{111} := A_{110} - A_{111} \end{array}$$

  3) $$\begin{array}{l|l} C_{000} := B_{000} + B_{010} & C_{001} := B_{001} + B_{011} \\ \hline C_{010} := B_{000} - B_{010} & C_{011} := B_{001} - B_{011} \end{array}$$, and
  $$\begin{array}{l|l} C_{100} := B_{100} + B_{110} & C_{101} := B_{101} + B_{111} \\ \hline C_{110} := B_{100} - B_{110} & C_{111} := B_{101} - B_{111} \end{array}$$

  4) $$\begin{array}{l|l} D_{000} := C_{000} + C_{100} & D_{001} := C_{001} + C_{101} \\ \hline D_{010} := C_{010} + C_{110} & D_{011} := C_{011} + C_{111} \end{array}$$, and
  $$\begin{array}{l|l} D_{100} := C_{000} - C_{100} & D_{101} := C_{001} - C_{101} \\ \hline D_{110} := C_{010} - C_{110} & D_{111} := C_{011} - C_{111} \end{array}$$

[M] GPU – Tensors contractions

Canales

- Same contractions as in the MPI project, but here the input is a list with thousands of pairs of (small) cubic tensors A and B. Goal: compute the corresponding R's using a GPU.
- $A, B \in \mathbb{R}^{n \times n \times n}$, with $n \lesssim 100$
- Two nodes: A and B are stored in P_0 and P_1, respectively. Assume each node has enough memory to store only $n^3 + b * n^2$ doubles.
- Storage by columns and by face.

[M] MPI – Matrix Transposition

- $M \in \mathbb{R}^{n \times n}$, distributed across p nodes; $1000 \lesssim n \lesssim 20000$, $4 \leq p$.
- Consider two cases: 1D block cyclic distribution, and either 2D block cyclic distribution or 2D elemental distribution;
- Implement the matrix transposition.
- Fix the problem size, and study the scalability with the number of nodes.

[M] MPI – Matrix Transposition

[M] GPU – Sequence of FFT-like operations

- Input: thousands of cubes $\in \mathbb{R}^{n \times n \times n}$, stored in main memory; $n \lesssim 150$.
- For each cube, compute D.
  Store it in a file, or overwrite the input (in main memory).

[E] CPU – Eigenvectors matching

- Track the evolution of the eigenvectors in a sequence of generalized eigenproblems
- Your language of choice; (Matlab makes your life easy)
- Test different approaches; measure the individual/total variation
- Input matrices (careful! 4.6GBs):
  `http://www.aices.rwth-aachen.de:8080/~pauldj/courses/hpmc-11/Matrices.zip`